NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

HUMAN INTERACTION WITHIN
A VIRTUAL ENVIRONMENT
FOR SHIPBOARD TRAINING

by

James E. O'Byrne

September 1995

Thesis Advisor: Michael J. Zyda
Thesis Co-Advisor: John S. Falby

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE		*Form Approved* *OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time reviewing instructions, searching existing data sources gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE September 1995	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE HUMAN INTERACTION WITHIN A VIRTUAL ENVIRONMENT FOR SHIPBOARD TRAINING	5. FUNDING NUMBERS
6. AUTHOR(S) O'Byrne, James Edward	

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The problem addressed by this research is that the existing Damage Control Virtual Environment Trainer (DC VET) simulator is deficient in the capability of presenting information about the environment. First, it lacks facilities for explaining the functions of engineroom equipment such as; boilers, pumps, gauges, switches and valves. Second, it lacks a facility to instruct users by a "Virtual Instructor/Guide".

The approach taken was to refine the DC VET simulator and increase its immersive interactive shipboard training capability. This was accomplished using the Jack Motion Library to create articulated human-form entities. Next, scripted actions of a human-form instructor/guide, combined with audio feedback in the form of sound effects and digitized speech via hyper-text links, instruct the novice user.

The result of this thesis was the implementation of a virtual ship model where networked users are represented as articulated humans who can see and hear engineering casualties. Actions of an instructor/guide may be scripted by a non-programmer. Using scripts DC VET has the ability to teach a novice the basic functions of boilers, pumps, gauges, switches and valves. Users can also interact with other networked users and discover functions of boilers and pumps in the engineroom by tagging the equipment. It is possible for the novice sailor to learn basic functions of engineering equipment before arriving at his ship.

14. SUBJECT TERMS Walkthrough, Virtual Environment, Training, Navy, Network, Simulation, 3-D Modeling, Jack Motion Library	15. NUMBER OF PAGES
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

HUMAN INTERACTION WITHIN
A VIRTUAL ENVIRONMENT
FOR SHIPBOARD TRAINING

James Edward O'Byrne
Lieutenant, United States Navy
B.S., State University of New York, 1987

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

September 1995

Author: _____
James E. O'Byrne

Approved by: _____
Michael J. Zyda, Thesis Advisor

_____
John S. Falby, Co-Thesis Advisor

_____
Ted Lewis, Chairman
Department of Computer Science

ABSTRACT

The problem addressed by this research is that the existing Damage Control Virtual Environment Trainer (DC VET) simulator is deficient in the capability of presenting information about the environment. First, it lacks facilities for explaining the functions of engineroom equipment such as; boilers, pumps, gauges, switches and valves. Second, it lacks a facility to instruct users by a "Virtual Instructor/Guide".

The approach taken was to refine the DC VET simulator and increase its immersive interactive shipboard training capability. This was accomplished using the Jack Motion Library to create articulated human-form entities. Next, scripted actions of a human-form instructor/guide, combined with audio feedback in the form of sound effects and digitized speech via hyper-text links, instruct the novice user.

The result of this thesis was the implementation of a virtual ship model where networked users are represented as articulated humans who can see and hear engineering casualties. Actions of an instructor/guide may be scripted by a non-programmer. Using scripts DC VET has the ability to teach a novice the basic functions of boilers, pumps, gauges, switches and valves. Users can also interact with other networked users and discover functions of boilers and pumps in the engineroom by tagging the equipment. It is possible for the novice sailor to learn basic functions of engineering equipment before arriving at his ship.

TABLE OF CONTENTS

x

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I. INTRODUCTION

A. BACKGROUND

The final product application developed from this thesis is called the Damage Control Virtual Environment Trainer or DC VET. DC VET is the implementation of an interactive networked real-time virtual environment for moving about an entire ship for training. Other related work has been architectural walkthroughs of large buildings and sub-sections of ship models [AIRE90][BROOK92][FUNK94]. However, more research needs to be conducted to solve the inherent problems involved with creating a world where humans can react.

Walkthrough environments are much different from other kinds of virtual worlds since the view that is seen in the world is limited in sight. In these virtual worlds, aircraft and ground vehicles move about a large terrain at varying speeds, some being extremely fast. Objects like trees, mountains, buildings, and other vehicles can be seen for miles as long as they are in the line-of-sight. In a flight simulation, the designer is not as concerned about flying through objects that are attached to the ground. However, with walkthrough environments one needs to avoid walking through walls, furniture and the like. A walk-through's view is typically much less than 100 feet and within that small area there are many different objects to view. A model of a house or ship can become quite extensive. However, efficient ways of traversing through the virtual world in real-time and in a real-istic manner must be found. Not all of the same model construction and rendering tech-niques apply to a walkthrough environment as do for a battle simulation on a desert terrain.

The DC VET is designed to enable the novice sailor to acquire ship familiarization by allowing him to move about the ship model in a realistic fashion. By visiting key points of interest within the ship model, the sailor can associate the virtual world later with the real world. The sailor can also review fire-fighting techniques when fighting a simulated fire in the engineroom compartment. The virtual environment is designed to instruct the sailor and allow him to review his skills. Team training is also an important part of Damage Control Training. With the use of Distributed Interactive Simulation, multiple people

1

participate in team training over an Internet connection [LOCK94]. One can participate actively or be a silent observer and watch others react in the simulation. The limit to the number of players is based on the limitation of the participant's workstation.

The navy has used simulators and mock-up trainers for decades to train naval personnel. The Damage Control Virtual Environment Trainer was conceived to be the next device to complement the navy's existing training. It can help to reduce training costs and also train people in areas that could not have been done before by eliminating dangers to personnel and equipment.

B. MOTIVATION

1. Navy's Training

Naval training of fleet sailors has traditionally started with classroom instruction, followed by simulators and hands on training once aboard ship. Classroom instruction can have many styles ranging from classroom training of a group, to training with a video presentation. Often after a level of instruction is completed, there is mock-up training in a controlled environment. It is this training that virtual environments can complement and enhance.

Most trainers are man-power intensive and require many dollars to maintain. It becomes too expensive, and there is often not enough time, to train sailors using these trainers. So the sailor receives a limited feeling of what the real task is like and does not necessarily maximize skills that could have been learned before reaching the fleet. Virtual reality can bridge that deficiency before the sailor reaches the fleet. It can also be used to continue training after reporting aboard ship. A sailor can receive training in the virtual environment and learn the necessary procedures to complete a task in spite of never really being there. He can then go to a facility trainer with those learned tasks and emphasize what he can physically only do in a mock-up trainer. Attention shifts from learning procedures in an unfamiliar environment (trainer) to accomplishing tasks in a reasonable amount of time in a familiar setting.

The second point to be made about virtual reality is that it can assist naval training with shipboard hands-on training. A virtual environment can assist sailors in becoming familiar and comfortable with the operation of equipment. We can build a Virtual Reality Laboratory onboard where sailors can train with various equipment. Procedures for PMS (Planned Maintenance System) that are not accomplished often can be reviewed. A technician can practice taking apart a very expensive weapons system for maintenance at sea, even if the physical maintenance can only be accomplished in-port. The Virtual Reality Laboratory augments his technical manuals and other references providing an interactive visualization of the task at hand as if working on the actual equipment itself. In theory this could reduce damage to a system caused by inexperienced technicians. Today many companies are looking at virtual reality to assist in these tasks [ADAM95].

2. Virtual Environments

The navy is not the first to consider virtual environments as a means of training personnel. Many companies are looking at virtual reality to train company personnel. With virtual reality, companies can save money and time and also train people in areas they could not consider before. As technology has progressed in the past two decades, researchers have been developing many ways to immerse a person into the virtual world. Companies like Motorola have used virtual reality in their training. They have not only found that they can save money, but have also increased worker's skills [ADAM95]. The three groups in Motorola's evaluation were traditional laboratory instruction, virtual reality with a computer monitor and the other with a Head Mounted Display (HMD). Motorola observed that the virtual reality group using a HMD reduced operator error rates drastically compared to the other groups which used a monitor for virtual reality and the traditional instruction. The reason given by the operators who performed better was that they felt a sense of presence in the virtual environment. They remembered what happened in the world simply because they experienced it.

C. OBJECTIVE

The objective of this thesis is to develop a real-time networked interactive virtual environment for training of shipboard naval personnel, particularly Damage Control. The virtual environment represents a real ship, as practical as possible, and tailored to simulate real life casualties.   In this virtual environment people can gain familiarization and learn skills which will enhance current training methods. In the construction of the virtual environment fidelity, realistic interactions and level of detail have been balanced against frame rate while giving the user a feeling of immersion in the world. Experience is gained from operating and interacting within the virtual environment.

The continuing research accomplished here has taken the current virtual environment and enhanced many of its features. Realism has been enhanced by increasing the interaction and responses perceived by the user. This includes articulation of a human model and the ability to manipulate objects in the world. Sounds are also used to assist in perceiving interactions. Refinements to the Collision Detection algorithm employed are implemented. All modifications must achieve enough realism to make the user feel the virtual environment.

This thesis focuses on Damage Control Training using a virtual environment to augment reality. The virtual environment is designed to train a fire team or even the Damage Control Assistant (DCA) in damage control. For example, a virtual environment can assist the DCA who needs to prepare for a mass conflagration. The DCA must design a scenario and carry it out. This can take many hours of planning and an incredible amount of ship's force man hours while training key personnel. Since practicing procedures and tactics are required, a networked virtual environment with just key players can accomplish this. Training can start with a dozen people participating so the whole ship does not need to be involved. They can keep training while not interfering with other shipboard activities until the training concepts become second nature. When all the key players are well versed then an all hands training scenario with the ship's compliment can be implemented. If the reaction time of key players has improved, then the emphasis of training on more rudimentary skills can be done during the all hands training. The use of a virtual environment in

this example can be applied to many other training evolutions. The objective is to reduce overall training time and maximize the use of limited personnel resources. If this example has eliminated just one mass conflagration drills, then virtual reality has saved hundreds of man hours.

D. CHAPTER SUMMARY

The remainder of the thesis is broken down as follows:

- Chapter II presents an overview of background information about the current computer system used to design the walkthrough. It includes a discussion of the hardware and software used. This chapter also provides an overview of the design considerations (and trade-offs) employed in building the ship virtual environment trainer.

- Chapter III discusses the integration of the *Jack*® Motion Library software from University of Pennsylvania.

- Chapter IV describes embedded training aids and scripted demonstration for training.

- Chapter V discusses how users at different workstations are networked together to interact in the same virtual environment.

- Chapter VI discusses in detail Collision Detection and object manipulation.

- Chapter VII introduces the user interface used in the virtual environment and describes how sounds are used to enhance the realism of the simulation.

- Chapter VIII provides a final discussion of the results of this thesis and describes follow-on work to be accomplished.

II. OVERVIEW

A. PREVIOUS WORK

Virtual worlds were devised to bridge the real world and graphical representations of the world. There have been other walkthroughs of architectural models, mostly by university research groups working on fundamental efforts to minimize polygon flow to the graphics pipeline. This research in computer modeling of walkthroughs has been accomplished by Airey and Brooks at the University of North Carolina (UNC) [AIRE90] [BROO92] and Funkhouser, Sequin, Khorramabadi and Teller at the University of California Berkeley (UC Berkeley) [FUNK94]. UNC currently has a model of Professor Brooks' house as a test bed which is comprised of 367,000 radiositized triangles. Recent work has been aimed at improving the Potentially Visible Set (PVS) geometry by employing a simpler less exact method which is determined dynamically at render time. This idea replaces the exact PVS algorithm which is performed at preprocessing time. The new concept employs bounding boxes instead of general convex regions. This renders each object once for every portal sequence, but allows the primitive-level clipping scheme to visit objects more than once, yet have no portion of the primitive rendered more than once [LUEB95].

UC Berkeley has created a practical approach to the manipulation of objects in a virtual environment. Berkeley's project, called "WALKEDIT", is a model of Berkeley Soda Hall which allows the user to manipulate 3D objects using a 2D I/O device. Objects which are manipulated are intelligently relocated based on the associated objects around them. It provides automatic implicit grouping which allows, for example, the user to move a book from the floor to a stack of books on a desk rather intuitively. The designers of "WALKEDIT" were able to design a cluttered office with furniture, books, cups, etc. in less than ten minutes [BUKO95].

The Naval Research Laboratory (NRL) in Washington D.C. is also currently engaged in using virtual environments for the study of damage control training. NRL's model is the engine room of the Ex-USS Shadwell. The model consists of about 500 polygons.

Immersion into the world is by means of a Head Mounted Display (HMD) or Binocular Omni-Oriented Monitor (BOOM), and mouse for speed input. Environmental effects like smoke and fire are added for realism during the training [NRL95].

Walkthrough environments, like the two mentioned above, still lack interaction between the user and other human entities. The need for representing other human figures and how they articulate is important for training people. The University of Pennsylvania has done excellent research in the development of human agents within the virtual environment [GRAN94]. They have created a *Jack*® Motion Library which is currently used with the Naval Postgraduate School's Networked Virtual World (NPSNET) [MACE95].

Developing graphical systems that represent true reality in real time is not achievable today because of the enormous number of objects that have to be created in the computer environment. Photographic resolution along with real time interaction is off in the future. For now we can create virtual worlds that represent reality. It is much the same with animation. To provide a real time drawing of a world we need a minimum refresh rate of 12 frames/second much like the old silent movies [BROO92]. This speed is adequate for immersion into the virtual environment but our goal is 15 frames per second. The modeling of the real world is represented in the virtual world by the use of polygons. It has been estimated that visual reality as envisioned in a computer graphic image consists of eighty million polygons per picture [CATM84]. However, current systems can only render about one million polygons per second. When rendering 15 frames per second, a real world image would take about 1200 million polygons per second. As the numbers indicate, graphical machines are not close to representing real worlds soon.

This thesis expands upon the recent shipboard walkthrough trainer work accomplished at the Naval Postgraduate School (NPS). The basic design of DC VET was developed by Tony King and Perry McDowell as their thesis [KING95]. They used an existing model of a proposed roll-on/roll-off commercial ship, the Antares, which was built for Naval Sea Systems (NAVSEA) by Advanced Marine Vehicles. The base model was created using MultiGen modeling software. The interior sections of the ship model were remodeled

to represent a naval vessel. This ship model was loaded into a Silicon Graphics (SGI) IRIS Performer based application. DC VET loads the model and allows the user to move about the ship. During the design, concepts of Potentially Visible Sets (PVS) and Level of Detail (LOD) were utilized to reduce the size of the model to be rendered before reaching the graphics pipe line [FUNK94]. To add realism into the ship model, a Collision Detection algorithm that uses Performer's line segments determines if the person has collided with any other objects in the virtual world.

The existing ship model of the Antares is used as the training platform in which the user can associate embedded world objects to immerse himself into the environment. The addition of sounds is used to enhance the immersion into the virtual environment. The introduction of an articulated human called *Jack* is employed to represent players in the networked virtual environment [GRAN94A]. This allows users to interact in an environment with each other in a manner similar to that in the real world.

B. HARDWARE

The system currently used to construct this trainer is the Silicon Graphics Onyx Reality Engine 2 (RE2). The RE2 incorporates a multiprocessing architecture, PowerPath2, to combine up to 24 parallel processors based upon the MIPS R4400 RISC CPU, which operates at 150 MHz. The I/O bandwidth is rated at 1.2 GB/second to and from memory, with support for the VME64 64-bit bus, operating at 50 MB/second. The RE2 is rated at 2M flat triangles/second and 900K textured pixels [NATI94]. NPS's version of the Onyx RE2 is a four processor 128 megabyte machine. The Silicon Graphics Power Series Reality Engine I is also used. This graphics machine has four R3000 40 MHz processors, a single RM4 board and an integral SCSI controller. It is rated at 1.1M flat triangles/second and 160M textured pixels. The Power Series Reality Engine I has been able to run DC VET at frame rates of 15 frames/second on average. The Onyx RE2 DC VET frame rate is 20 frames/second on average. The frame rate may decrease as the complexity of the project grows. However, to ensure real-time interaction, complexity is limited to maintain a minimum

frame rate of 15 frames/second. The intent is to develop a relatively low cost system for naval ships and shore facilities. Silicon Graphics has released a new system, called Indigo² Impact, which has the capability of the Power Series Reality Engine I for about $50,000 [ERTE95]. This new system would be ideal for shipboard deployment simply because of its real-time graphics capabilities, low material support and overall cost.

C. SOFTWARE

1. Rendering Software

To support the objective of real time graphics simulation, IRIS Performer was chosen to create the rendering software. Performer is an Application Programming Interface (API) whose architecture is designed to support high performance, multi-processed graphics applications; especially, visual simulations, virtual reality and real-time three-dimensional graphics. Performer is based upon a hierarchical database, an example of which is shown in Figure 1. Performer culls the database so that only PVS geometry is sent to the
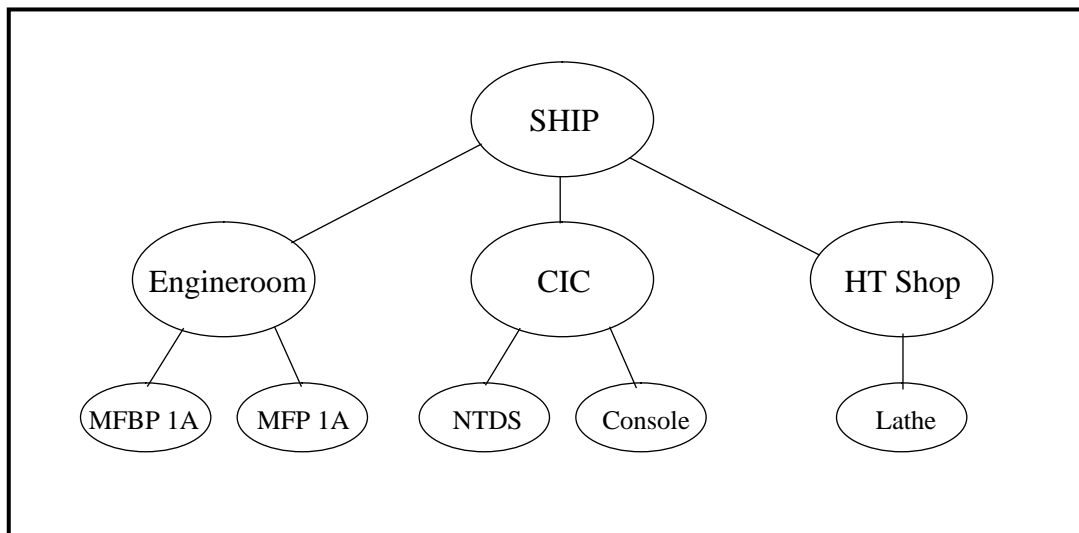
Figure 1: Partial Ship Database Hierarchy

graphics pipeline. It also provides fast intersection testing through database traversals, and performs LOD management of objects. The most significant feature about Performer is

that the cull, draw and application processes can run in parallel, greatly increasing the speed of the program [ROHL94]. Performer is capable of reading 14 different types of databases into an application.

2. Modeling Software

The current ship model for DC VET was designed using the MultiGen Modeling Tool produced by Software Systems. MultiGen is a comprehensive format that can represent nearly all of IRIS Performer's advanced concepts, including object hierarchy, instancing, LOD, light point specification, texture mapping and material property specification.

3. Operating System

The current operating system used in the Naval Postgraduate School Graphics and Video Laboratory is Silicon Graphics IRIX 5.3. All new developments and evaluation of the shipboard walkthrough environment is performed using it.

4. Software Libraries

a. *Jack*® Motion Library

The University of Pennsylvania's *Jack*® Motion Library is used to create an articulate human entity [GRAN94A]. Using their library software in conjunction with the existing virtual environment application, we can create a human who appears to act in a realistic manner. The human model has the ability to move arms and legs to simulate motion and can also be designed to change posture. Previously we could only display a human icon and float him about the Antares model.

b. Distributed Interactive Simulation (DIS) Protocol

The DIS protocol [IST93] is used to ensure compatibility with other NPS applications. Currently DC VET is designed to be a network simulation. Multiple players can communicate while running the same application and can train together in the same virtual environment. Users need not be in the same physical locale, just anywhere there is an

11

Internet connection. Planned design considerations include incorporating the DC VET features into NPSNET.

D. DESIGN CONSIDERATIONS

5. Real Time Rendering

a. Performance

In order to maintain immersion of the user into the virtual environment, one must ensure that the overall performance of the application is in real time. A high frame rate is desirable since slow and variable rates tend to reduce the illusion of reality, particularly when interaction is between the application and the user. Long lag times between an action and a perceived response must be avoided. Methods of PVS, collision detection, and LOD decrease the amount of the model to be rendered and increase overall performance remarkably [TELL91]. Also, a hierarchical database model and limiting the polygon count in the model design improves speed and performance.

In the Antares model the use of LOD did not play as important a role as PVS in polygon reduction. LOD works best where there is little obscuration of other objects, these objects are visible for long distances, and they are hardly recognizable at long distances. In the ship model most compartments are at most 20 meters long with the exception of the engine room, and objects are obscured by room boundaries. Also, an object is rarely visible at long distances from the viewer. Thus, the PVS algorithm performed better than LOD in reducing rendered polygon count.

In the construction of the ship model objects placed into the world are considered carefully. A scene from a ship would have considerably more objects than a house or building. Since the goal was not to build a precise model of a ship but a suitable trainer for the transfer of knowledge of damage control skills, many intricate details of a ship's interior were deemed unnecessary. For instance, the lights, piping and wiring in the overhead are not modeled. It is not essential to create every object since the inclusion of many extra objects adds to the size and complexity of the model. This increases the polygon count and

decreases the frame rate of the program. Once below 10 frames per second, the immersion of the user into the virtual environment is lost. Some excellent work in real-time shading [LAST95] and anti-aliasing has been done. However, such features are not incorporated into this project since the design of the ship model is for training purposes.

b. Immersion

To gain the most training out of the walkthrough virtual environment, the user must believe he is a part of the world by being immersed into it. One of the best devices in providing this illusion is the HMD. It gives the user a wide field of view, much like one's eyes. With the aid of tracking devices, the HMD provides to the graphics machine the direction in which the user is viewing. This is immediately translated and allows the user to view the virtual environment much the same way he normally sees the real world. However, there still needs to be some other input device to resolve the speed the user is moving and any choice of interaction with objects or other entities. Currently a three button mouse is used to perform this interface. Perhaps a data glove may be a better choice for input, but that has not been used because of its cost and mean time to failure of the device. Voice recognition is another consideration which we are researching. It can be used for some limited control functions such as moving and stopping, opening and closing doors and manipulating objects. If recognition software can reliably understand rudimentary commands of a limited set, then this may be the best choice for an input device. In addition to giving commands, the user should hear the environment. As he traverses the model, he should encounter sounds that are realistic and give him guidance. Sounds and sound effects have been embedded into DC VET to enhance realism.

NPSNET is also designed to operate with a head mounted display infantry man, IPORT. IPORT immerses the soldier into the virtual world and uses a force feedback pedal device to make the soldier feel the terrain as he moves about. Immersion of the user into the virtual environment is key to maximizing training.

c. Database Model

A hierarchical design of a walkthrough model is essential in obtaining real time performance during rendering. A large model will have many thousands of polygons to create for the scene so that a user can interact. The designer must consider what type of graphical machine is to be used to display the model in real time. In this thesis design, trading less realism for rendering with a high frame rate was implemented. A ship in many ways is a more complex model to render than the inside of a building or house. A ship has many doors, passage ways, and objects hanging from the bulkhead and overhead. Most of all when modeling a ship there are ventilation ducts, piping and wiring everywhere on a ship. All the fine details can not be currently rendered in real time because of the sheer number of polygons in the scene required to do so. In building a virtual environment decisions must be made on what is important for training. Certain objects must be represented in order to make one believe he is on a ship, but too many objects represented may hinder the frame rate. The final design decisions come down to what the virtual world is to be utilized for and what are the necessary requirements to achieve the desired level of training.

The Antares ship model represents a ship but in no means is a precise model. It strikes a balance of flat shaded polygons and vivid textured polygons to create realism while maintaining real time performance. Objects for the virtual world are constructed on a needed basis. Consideration for training minimums and rendering of the scene are continually evaluated during the design.

d. Training

Training, which is the goal of the Damage Control Virtual Environment Trainer, must be considered throughout the construction of the virtual environment. In the virtual environment trainer, fire fighting tools and training objects are placed around the scene to assist the user in learning fire fighting techniques. The objects in the virtual environments are icons which represent the real item. The environment is constructed to look real enough for a user to know instinctively what things are. Also, with the use of

14

hyper-text and sounds, we can help the user learn about objects he is not sure of. It would be unrealistic to expect even a new sailor onboard a real ship to know what every piece of equipment is and what purpose it serves. It is the same in the virtual world, but he can learn what things are via the trainer and not have to question other people.

E.   CONSTRUCTING A VIRTUAL SHIP ENVIRONMENT

An actual model of a naval vessel was highly desired for this project. Building a model from the blueprints of a commissioned naval vessel proved to be more work than a few people at a laboratory could do in a few months. The next thought was to obtain an AutoCAD file of a naval vessel from shipbuilders of naval surface ships. It was discovered during a survey that each contracted ship builder, even for the same exact class of ship, had a different format of AutoCAD. An importer would have to filter out the unnecessary data and retain the needed 3D data for the walkthrough program. However, an AutoCAD file of an actual naval vessel was not available in time for this research. It was decided to use the Antares model from NAVSEA in Washington D.C. for experimental purposes. The original Antares model, built using a MultiGen flight format, comprised about 2,000 polygons. When modified internally to represent a naval vessel it was over 22,000 polygons.

The model is constructed using Software Systems MultiGen modeling tool, which is an off the shelf product. The advantage of the MultiGen modeling tool is the ability to design a model in a hierarchical fashion allowing for simpler management with the IRIS Performer rendering software. MultiGen's hierarchical structure uses a directed acyclic graph (DAG) to store the visual database. A partial hierarchical view of a ship model is provided in Figure 2.

MultiGen provides switch nodes to take advantage of Level of Detail (LOD) implementation instancing of exact multiple objects to reduce the model size. Textures are attached to polygons and stored within the model improving the illusion of realism. MultiGen's greatest feature is the ability to embed non-visual information. This feature is paramount to the success of the virtual environment trainer because it incorporates the use of hypertext displays, PVS, collision detection, and the orchestration of objects.
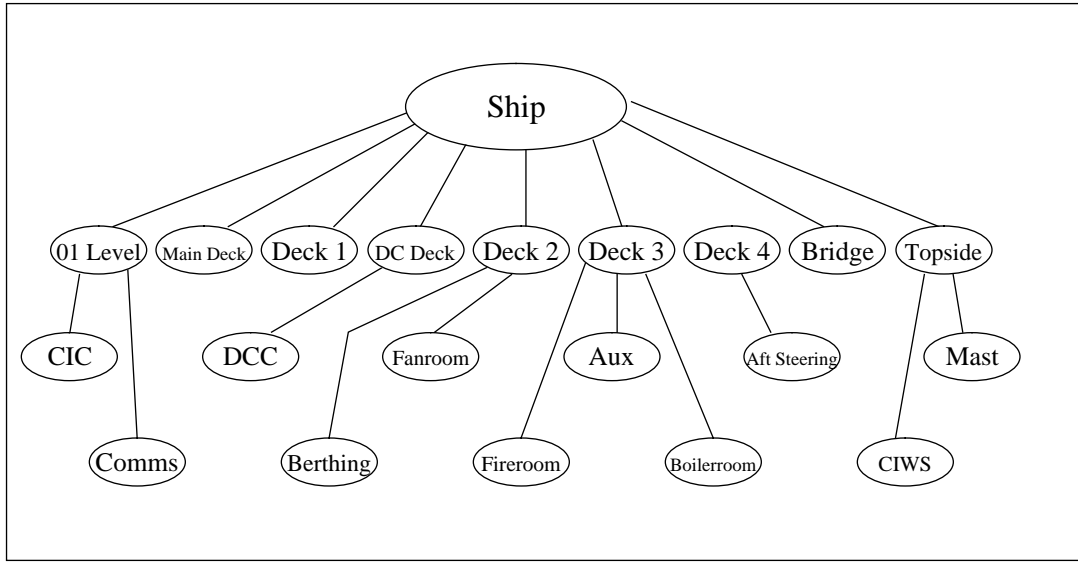
Figure 2: Spatially Partitioned Database Organization

The characteristics of the object are stored within the model. When designing the model, objects which have motion are tagged and called Degree of Freedom (DOF) nodes. The articulation of the node is constrained in the model database. When designing a door, for instance, the rotation of the door node is limited to 90 degrees about the Z axis. To simulate damage to a bulkhead of a ship from a fire, the normal texture is swapped out in stages to illustrate the progress of the damage. These animation textures are incorporated into a switch node and are swapped on the fly during a damage control training scenario. All the embedded data is stored within the model identified by the Performer application during the initialization phase. The nodes are read by the application then categorized by their type. This allows the flexibility of making changes and additions to the model without having to modify code in the Performer application. For instance, if a deck is deleted, or another instance of an object in the model is added, such as a door or fire station, no additional programming requirements are needed. However, if the new type of object is something which presently does not exist in the model, such as a vertical hatch, it would have to be coded in the Performer application to handle the object's characteristics.

16

The Antares model chosen for this project was not a naval vessel, but it was still more than adequate for testing and evaluating the power of training within a virtual environment. For actual naval training a model of an actual naval ship from bow to stern is preferred. Modeling of a naval vessel will be follow on work.

III. *JACK*® MOTION LIBRARY

The *Jack*® Motion Library (*Jack* ML) of the University of Pennsylvania is incorporated into DC VET to replace the previous human icon representation. The older human icon was a 3D figure of a sailor, see Figure 3, which had no motion. The icon could be moved about the ship and rotated to indicate the position of a networked user. However, this failed to truly represent the user's motion and reduced the illusion of immersion. With *Jack* ML the body can be articulated so one can see the head, legs and arms move as a networked user participates in the environment. This gives the sense of realistic movement allowing a person to react better to the visual gestures of other users.

Figure 3: Previous Graphical Representation of a Sailor

A. SOFTWARE LIBRARY

Jack is a general purpose, interactive environment for manipulating articulated geometric human figures in a 3D interactive environment. *Jack* has a rich notion for building articulated figures with revolute and prismatic joints. *Jack* ML is a general purpose constraint engine that uses an iterative inverse kinematics procedure with high degree of freedom (DOF) joint chains [GRAN94A]. The *Jack* ML animates the human figure by transitioning from one posture to another or locomoting in a cyclical posture change. *Jack* ML then passes the joint angles back to DC VET for animation by the IRIS Performer run-time articulated database of human geometry. There are two *Jack* models to choose from, a high and low resolution. At NPS we use the low resolution DI figure because of NPSNET real time frame rate requirements. This low resolution version emulates the high resolution version in most details, except it has no fingers (fingers and palm are a single element), no spine, no eyeballs, and no clavicle. The low resolution *Jack* is a 478 polygon model which is comprised of 23 joints having a total of 50 degrees of freedom, see Table 1 [GRAN94B].

	Soldier.fig	**DI.fig**
POLYGONS	2410	478
EDGES	4472	773
NODES	2510	327
SEGMENTS	69	24
SITES	180	147
JOINTS	68	23
DOFs	134	50

Table 1: *Jack* Figure Characteristics

B. FUNCTIONALITY

DC VET's version of *Jack*, called BlueJack, is a sailor version of the DI figure. The networked players see each other as *Jack*, shown in Figure 4. *Jack* has the ability to display the motion of walking or running, head movement and hand motion. *Jack* ML operates

Figure 4: Jack As a Sailor

concurrently with the DC VET application as a separate process. This is an important point since both processes are running and executing different commands which are not always in sync with each other. Information has to be passed to the *Jack* ML via function calls. For DC VET the parameters for head rotation and elevation, appearance, velocity and heading are passed every cycle. However, hand motion is accomplished differently. A separate function within the *Jack* code checks to see what position *Jack's* hands should be at for the next *Jack* ML cycle. For DC VET to manipulate hand motion, hand signals have been established and an index for each possible signal is created. When a certain hand signal is

desired, a queue in DC VET is filled with the request. Next, when *Jack* ML is ready to execute that signal, it reads the queue, processes the request and executes the hand motion. The queue is cleared after the signal is completed and the next hand signal, if any, is processed. Without the use of a queue, the *Jack* ML may not receive the hand signal request before the next cycle and miss the desired hand motion.

Hand motions have to be calculated by breaking down the joints and moving each segment by the appropriate amount of degrees within a designated time span, see Figure 5.

Shoulder Joint (3 DOF)

Elbow Joint (1 DOF)

Shoulder DOFs
1) Up/Down
2) Swing Left/Right
3) Twist

Elbow DOFs
1) Up/Down
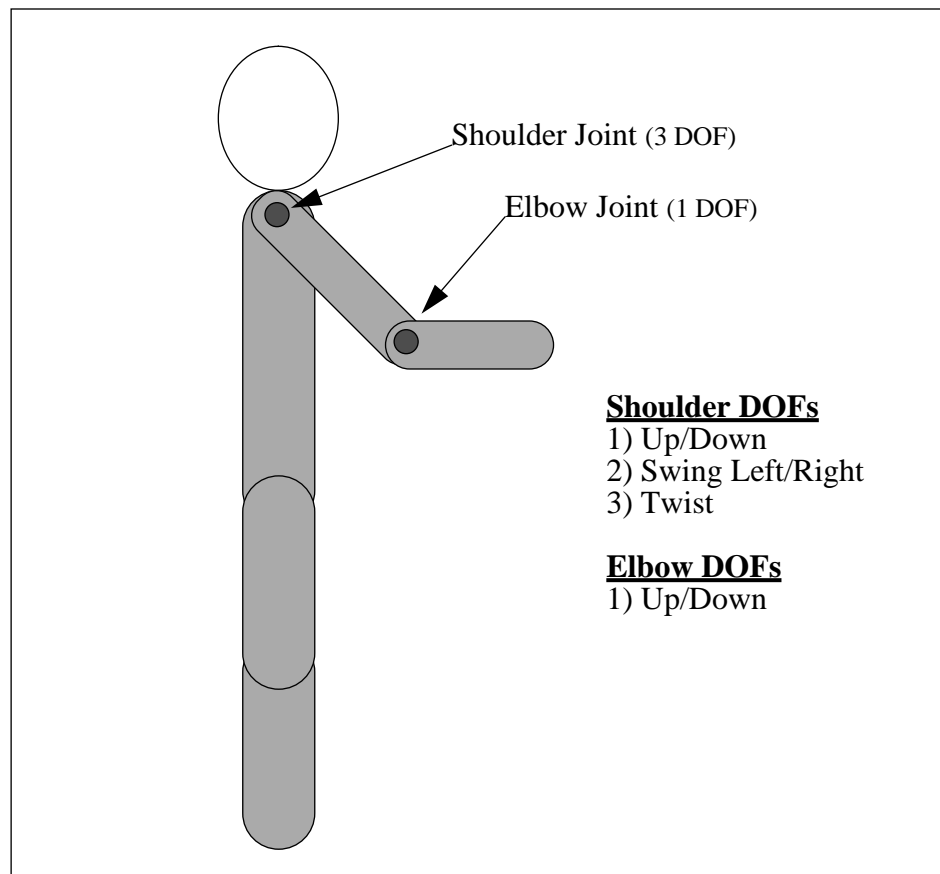
Figure 5: *Jack* Articulated Arm Joint Composition

The programmer must conceive all of the possible hand movements that he wants *Jack* to perform in the virtual environment. This requires a little pre-planning and calculating to create realistic motions. The motions are given names for each hand signal which are then

hard coded into a file called "HandSignals.C". Currently there are over two dozen hand motions written, including an arm swinging motion for walking.

Movement of the *Jack* DI figure throughout the environment is controlled by the DC VET application. DC VET controls the translation, dead reckoning, body rotation, and velocity while *Jack* ML performs the animation. DC VET controls these functions in order to control multiple entities while conducting collision detection. This also allows for the smooth swapping of clothing texture from a typical sailor uniform to a firefighting outfit. The initial incorporation of *Jack* ML into DC VET suffered from compatibility problems with texturing, removing of the rifle and initializing animation. A particularly frustrating problem arose in the initial animation sequence. In order to have any animation accomplished, the programmer must initialize *Jack* to a positive velocity. Starting with zero velocity at initialization meant that *Jack* would never be animated, even later when *Jack* is moving. But once all the functionality was operational, *Jack* ML was well worth the effort.

Jack also has the ability to change postures. Keep in mind *Jack* ML was originally designed for a soldier not a sailor. These postures are stand weapon stowed, stand weapon fire, kneel weapon stowed, kneel weapon fire, prone weapon stowed, prone weapon fire, crawl and dead. In DC VET the standing, kneeling and dead postures are used. During the loading of the *Jack* software, the weapon is removed from the tree hierarchy and the uniform texture for camouflage is replaced with a sailor work uniform. The rifle is obviously not needed for damage control training, but the option to have a Marine with a rifle onboard a ship is available.

Jack has made a difference in the representation of the human in DC VET. Players in the environment can see the direction a networked user is looking, just like a real person. A person can watch networked users reach for objects, open or close doors and valves, and perform hand gestures such as come follow me, see Figure 6. Because of *Jack's* human-like qualities, two other features where added into the environment: a physical body for the user himself, and an autonomous *Jack* that navigates about the environment. You can see your legs move when you walk and your hand reach to open a door. Once again, these add to the feeling of immersion. The use of autonomous *Jack* allows the DC VET application
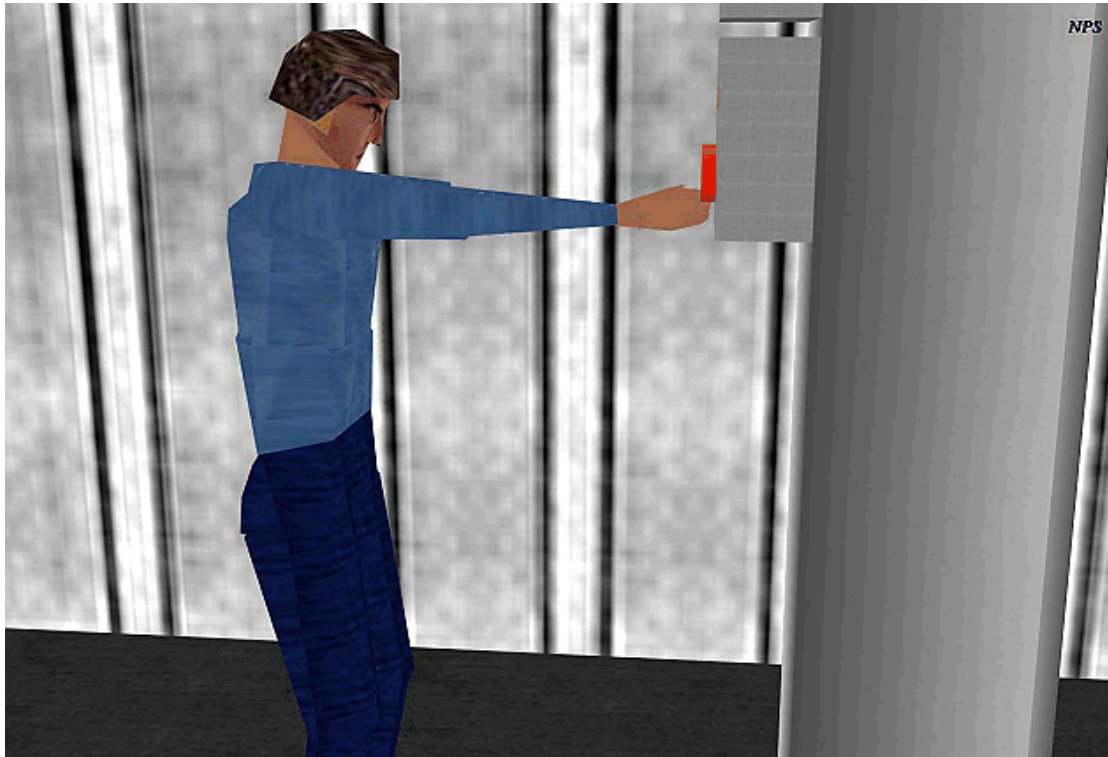
Figure 6: *Jack's* Hand Motion

to have a tutor inside the environment. A roving autonomous entity can teach you how to navigate about a ship and point out items of importance just like a person giving a guided tour.

The use of the low resolution *Jack* is implemented because DC VET is expected to have several people networked together with one or more autonomous Jacks operating at the same time. Each Jack consists of almost 500 polygons, and with several Jacks the polygon count can increase quickly while decreasing the frame rate. Operating DC VET with Jack ML proved to be capable of managing the graphical rendering load. Even on the slower Reality Engines, Power Series I, the application maintained good frame rates. With three networked stations and three autonomous Jacks, frame rates of 10 to 20 frames per second are achievable. Performance of the overall application is more than adequate to support immersion for training.

C. MOTION

Jack was incorporated to replace the original sailor representation and show human articulation in the virtual world. The desire to use the *Jack* figure in DC VET went beyond having *Jack* act as a networked entity. To gain familiarization with the shipboard environment, a guide was needed to help the user along. *Jack* has become the tour guide and instructor for new people in the DC VET. *Jack* can be scripted like a lesson plan to show and tell the user what his world is about. In order to have a *Jack* entity move in the world a method of moving him from point to point was developed. It is to be quick and easy to implement, and yet allows *Jack* to move in a natural manner.

When creating the autonomous *Jack*, the best motion path to travel is that of a human. To move *Jack* from one point (in three dimensions) to another, a 3D location is given along with the speed and final direction to be facing when he arrives there. An algorithm using trigonometry, to find the shortest path combined with a heuristic that uses a weighted average to orientate *Jack* in the proper direction was created. *Jack* does not whip around on the shortest course to the new place since humans turn progressively in the direction they want to go. Hence he turns at a rate appropriate to the desired final direction and shortest path. This is computed every frame cycle giving *Jack* a smoothing walking motion, see Figure 7. The walking algorithm orientates *Jack* to reach the desired location and heading even if he is facing the wrong direction as shown Figure 8. If *Jack* is given a path that did not have enough room to maneuver, *Jack* pivots his body to face the correct direction before starting on the next event leg.

Some of the paths computed by the algorithm are not the quickest path a human would walk. However, the algorithm does offer natural movement vice square facing movements like an army platoon.
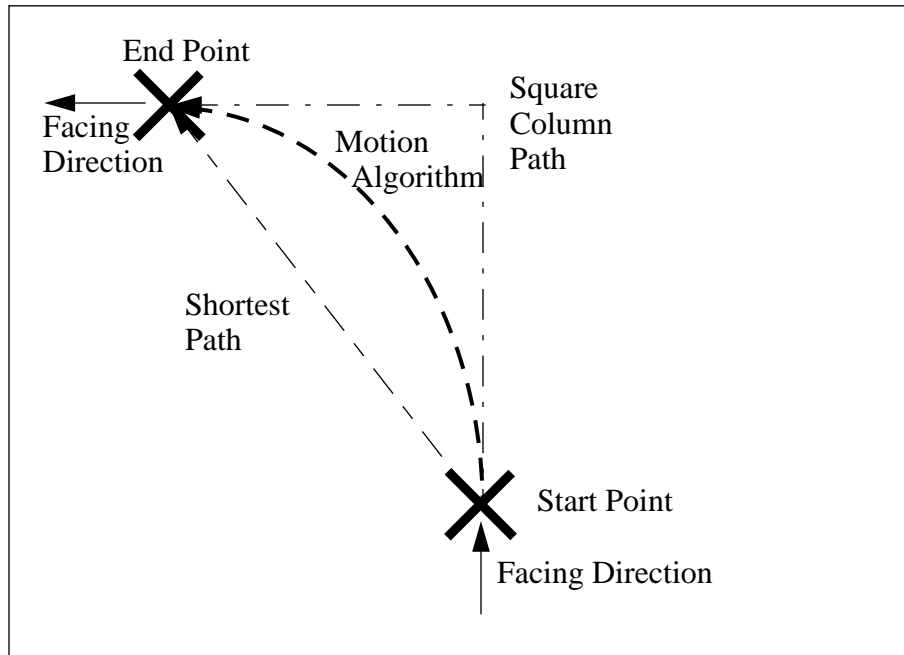
Figure 7: Walking Motion Path
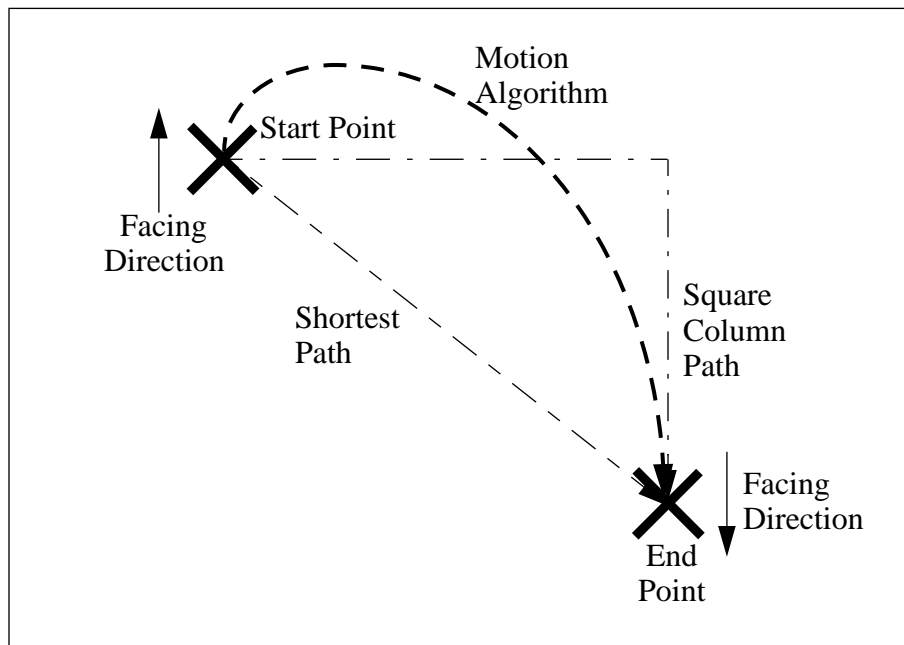
Figure 8: Opposite Direction Walking Motion Path

IV. LEARNING ENVIRONMENTS

A. TRAINING

The main focus of the original DC VET was to be a virtual environment trainer. Now, with the capability of creating three-dimensional worlds that look much like the real world and being able to navigate and interact in real time with them, new methods for teaching people have been unleashed. Virtual environments will be a new avenue for training much like video tapes which instruct people in any number of activities from aerobics to civilian flying. Today a person can go to a video store and rent a video to teach themselves civilian pilot skills for flying. Lessons range from take-offs and landings, navigation and in-flight emergencies. These tapes are an inexpensive way for a person to learn and prepare himself for the many hours of flight time required to qualify for a license. He can learn when he has time, and repeat lessons that necessitate additional instruction. Virtual environments proficiently accomplish instruction of these skills. People learn through experience and virtual environments are the next best thing to being there. DC VET's specific goal is to teach basic firefighting and engineering causality skills. It accomplishes this by allowing users to learn what things are in the environment by inquiring or simply observing articulated humans illustrating procedures in casualty control. However, DC VET has not reached the level of instruction necessary for teaching all the higher level firefighting skills. In time, advanced technology will enable even more realistic environments to be developed.

1. Methods of Learning

NPS has a networked environment, DC VET, where people can interact together, become familiar with their surroundings and learn procedures in fighting shipboard fires. A structured training program for instruction was desired for DC VET. It was imperative that the user could learn how to move about the virtual environment and interact with relative ease. The concept of using a guide to help the novice user in the world came about by

mimicking the real world where one would have a person to show him where things are and discuss their function This concept is similar to indoctrination periods that new employees go through. DC VET uses the autonomous *Jack* to guide the user through the ship to find new compartments and become familiar with the ship, see Figure 9. After the familiarization phase the user can progress and learn how to combat fires and engineering casualties

Figure 9: *Jack* Discussing a Pump's Operation

in the engineroom. He can watch *Jack* fight a fire and extinguish it, Figure 10, or cut off supply to a steam or fuel oil leak, Figure 11.

There are four options for the user to learn what is in the environment. First, watch *Jack* move through the ship and watch a totally guided tour. This is much like a video presentation. Second, the user can follow *Jack* through the ship and watch how he reacts. Third, he can be lead by *Jack* through the ship, but has the option to stop and explore objects of interest. Finally, he can just figure it out himself, at his own pace, by trial and error. Having the different options allows the user flexibility to learn at his own pace. He can
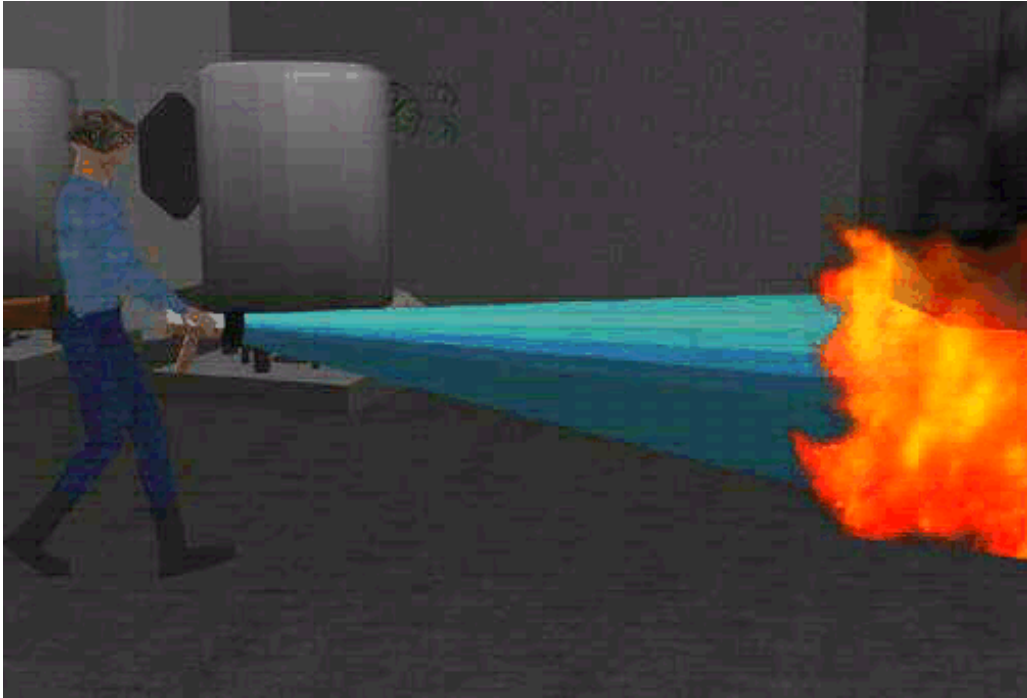
Figure 10: *Jack* Illustrating Fire Fighting Methods
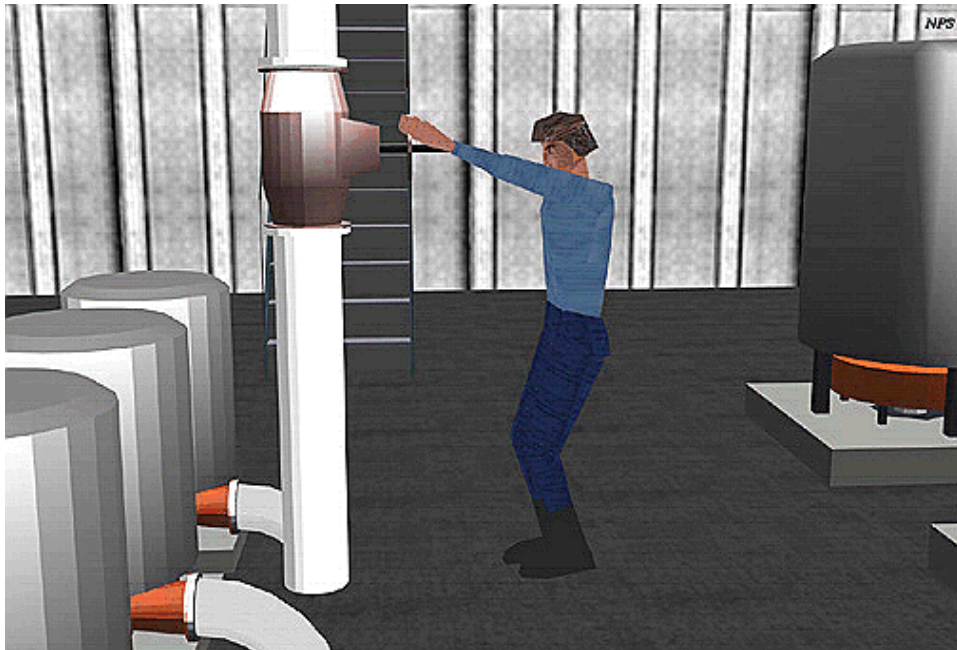
Figure 11: *Jack* Closing a Steam Valve

switch between modes, reverse or fast forward a scene and even re-start from the beginning at anytime.

With the autonomous *Jack* one can see a human figure guide him. It is intended that virtual trainers be capable of teaching the user about the environment vice having another individual constantly supervising. We have taken the instructor from the classroom and placed him into the virtual environment. The *Jack* ML can be used to give hand signals to point in a specific direction while simultaneously playing audio sound clips to present information. *Jack* can walk up to a Main Feed Pump, discuss its function in brief, point to the intake and discharge piping. When following *Jack*, he can notify you that you are going in the wrong direction or that he'll wait here for you.

2. Lesson Designing

The more complex the training and the more topics to be covered, the greater the amount of planning required. This is much like an instructor preparing an outline for class and deciding on the contents to be discussed. The better prepared the lecture the more beneficial the instruction. Training lessons are typed into a script data file by the lesson designer and can be played back at anytime while planning a lesson. The designer of a lesson can review where and what *Jack* has done and if the lesson meets expectations. When recording scripts, one has to take into consideration where (in three dimensions) *Jack* is planned to perform a task. Also, a numeric index is used to choose the different tasks presented in Table 2. Writing a scripted lesson is relatively easy since the planner just moves through the world and looks in the direction where he wants the events to happen. Then the positions are recorded with the event as he decides on the lesson.

Having a scripted lesson, which is not much different from an instructor's class outline, allows the individual flexibility in learning. This allows the user to progress through training lessons at his pace, repeat lessons that he desires, and skip other lessons that were simple to him. The training is dynamic and flexible to the user's requirements.

The user does not feel embarrassed or pressured to learn. The environment is well suited for the natural retention of knowledge and the learning of skills.

Scripted *Jack* Events			
Move	Pivot	Go Up/Down Stairs	Open/Close Doors
Play Sound-bites	Start Fire	Start Steam Leak	Pick Up Nozzle
Nozzle Water On/ Off	Perform Hand Signals	Activate Halon	Activate Ventilation
Open/Close Valves	Jump to Other Sections of the Ship	Wait/Pause	Display Text

Table 2: Autonomous *Jack* Instructions

Many of the features created for instructing autonomous *Jack* are choreographed with other events that affect the user and other networked entities. Therefore, consideration when designing new functionality requires rebuilding of the original DC VET applications. The redesigned version of DC VET uses similar control functions that manipulated networked entities to operate the new autonomous entities. To keep the networked simulation states congruent, any event that a networked or autonomous entity perform must be reflected in all networked simulations. The environment is now capable of supporting the user as an entity, networked and autonomous entities all at the same time.

DC VET is not at the point where we can break down equipment into smaller parts and train a mechanic to fix an engine. But future designs of virtual ship environments may allow a person to move around a ship and learn about the components of the Main Feed Pump. A person at some critical time will require the knowledge for casualty control training, maintenance or repairs. Training of this type is essential and needs to be expanded. The capability to train an engineer to operate and repair a piece of equipment at sea without an instructor is necessary. At sea we do not always have the expertise for every piece of equipment and can't wait weeks for the next port visit or technical representative to be

flown out. An onboard trainer can improve productivity, decrease equipment down time and save dollars.

B.   CORPORATE TRAINING

Motorola University was looking for a means of broadening the manufacturing training program [ADAM95]. They decided that virtual reality technologies would be a cost effective way to replace their training course for operating a robotic machinery line. Virtual Reality would provide portability of the training tool and allow for quick updates of changes on the production line. Motorola University tested their concept on the Advance Manufacturing Course of the Pager Robotic Assembly Facility. There were three groups of people for the evaluation. First, a trainer using a life size replica of the assembly line. Second, VR group using a desktop console. And lastly, a VR group using a HMD with tracker. They found that the VR group using a HMD with tracker tested much better than the other two groups. Comparison was recorded by the number of errors and missed steps made by each group. These results are presented in Table 3 [ADAM95]. Essentially the VR HMD with tracker did six times better than the laboratory or VR desktop groups.

TASK	MOCK-UP LAB	VR DESKTOP	VR HMD
Setup	13	14	1
Start-up	5	6	1
Running	0	0	1
Shutdown	6	4	1
Average	6	6	1

Table 3: Mistakes Observed in The Test Groups

Motorola concluded that the reason for virtual reality success in their project is that it is individually driven and people can practice at their own pace. The VR group with the HMD with tracker were totally immersed and focused in the training. The virtual environ-

ment was designed in six weeks using PC based software from Superscape. Detailed of modeling of the assembly line equipment was important as well as sound bites reflecting action in the environment. The operator of the environment can associate actual sounds in the virtual environment to the real assembly line later. Superscape does not provide as much functionality and capability as Performer and MultiGen used in the NPS graphics laboratory. DC VET uses the same basic concepts of Motorola and should be capable of realizing similar positive results.

C. SUMMARY

Continuing the enhancements of virtual trainers, DC VET needs to be transformed into an environment that can also support multimedia to supplement the realistic way of navigation and learning that already exists in the environment. Video and sound together are required to explain more articulated subjects until virtual environments are robust enough to simulate this themselves. For instance, a short video take can explain how to don an oxygen breathing apparatus (OBA), explain the refrigeration cycle, or illustrate how steam flows through the engineering plant. Multimedia in combination with the three-dimensional environment will enhance the training even more. A person later can walk up to an OBA on a ship and remember the video take on how to don the OBA. In this manner training can successfully be accomplished.

V. NETWORKED ENVIRONMENT

A. DIS COMMUNICATION PROTOCOL

A networked environment was essential to the design of the DC VET because most naval training is team training. It was necessary to have multiple players in the same environment interacting to solve the same problem. NPSNET has had great success with the Distributed Interactive Simulation (DIS) protocol which was used for this project as well [IST93]. All government built networked simulations must communicate via DIS [DOD92]. The DIS protocol uses packets of data called Protocol Data Units (PDUs) which are transmitted over the graphics laboratory's network. The entity state PDU is used to send the other players an entity's location in the model, direction and linear velocity, acceleration, posture, objects manipulated, and the scenario state. Each player is also assigned a unique entity number to organize PDU data. The use of linear velocity, acceleration, and dead reckoning parameters allows the program to plot the user's next position even if a PDU is lost.

Networking humans in an environment, instead of vehicles, presents different problems. Humans can change their orientation more quickly than vehicles and have many postures in a short period of time. This necessitates the propagation of more PDUs than a vehicle simulation in the same amount of time. A PDU is transmitted on a threshold basis, or if certain events occur. For instance, if the user moves a few meters, comes to a quick stop or moves an object. The timely transmission of a PDU is critical to the immersion, but the system should also not be overloaded with excessive PDU packets.

When in the network mode, the program constantly scans for entity PDUs throughout the walkthrough simulation. When a new player's PDU is received, it creates a new *Jack* sailor dynamically and translates him to the respective part of the ship model. This virtual environment can also operate while other DIS networked environments are operating without interference. The program discriminates other PDUs of other simulations via

a reference number located in the Exercise ID PDU subfield. A networked observer mode was also created so people can watch and learn what others are doing in the virtual environment trainer and not interfere with the training. The other players don't even know the observer is there. A final networked mode is a sound server which was created for the graphics reality engines without sound capability. A sound capable computer acts as the sound server, reading PDUs from the network and creating the sounds for one or more players.

Modifying DC VET to network motions represented by the *Jack* Motion Library required use of additional PDU subfields. The PDU type used in the NPSNET DIS protocol is shown in Table 4. Of all these PDU types to chose for DC VET, the Entity State PDU

Variable Length	Static Length
Entity State	Fire
Detention	Resupply Cancel
Service Request	Repair Complete
Resupply Offer	Repair Response
Resupply Received	Collision
Action Request	Create Entity
Action Response	Remove Entity
Data Query	Start/Resume
Set Data	Stop/Freeze
Data	Acknowledge
Event Report	Laser
Message	
Emission	
Transmitter	
Signal	
Receiver	

Table 4: Various NPSNET PDU Types

Field Size (Bits)	PDU FIelds	PDU Subfields
96	PDU Header	Protocol Version *8 bit enumeration*
		Exercise ID *8 bit unsigned integer*
		PDU Type *8 bit enumeration*
		Padding *8 bit unused*
		Time Stamp *32 bit unsigned integer*
		Length *16 bit unsigned*
		Padding *16 bit unused*
48	Entity ID	Site *16 bit unsigned integer*
		Application *16 bit unsigned integer*
		Entity *8 bit unsigned integer*
8	Force ID	*8 bit enumeration*
8	Articulated Parameters	*8 bit unsigned integer*
64	Entity Type	Entity Kind *8 bit enumeration*
		Domain *8 bit enumeration*
		Country *8 bit enumeration*
		Category *8 bit enumeration*
		Subcategory *8 bit enumeration*
		Specific *8 bit enumeration*
		Extra *8 bit enumeration*

Table 5: Entity State PDU

Field Size (Bits)	PDU FIelds	PDU Subfields
64	Alternate Entity Type	Entity Kind *8 bit enumeration*
		Domain *8 bit enumeration*
		Country *8 bit enumeration*
		Category *8 bit enumeration*
		Subcategory *8 bit enumeration*
		Specific *8 bit enumeration*
		Extra *8 bit enumeration*
96	Entity Linear Velocity	X - Component *32 bit floating point*
		Y - Component *32 bit floating point*
		Z - Component *32 bit floating point*
192	Entity Location	X - Component *32 bit floating point*
		Y - Component *32 bit floating point*
		Z - Component *32 bit floating point*
96	Entity Orientation	Psi *32 bit floating point*
		Theta *32 bit floating point*
		Psi *32 bit floating point*
32	Entity Appearance	*32 bit record of enumerations*
320	Dead Reckoning Parameters	Dead Reckon Algorithm *8 bit enumeration*
		Other Parameters *120 bits unused*
		Entity Linear Acceleration *3x32 bit floating point*
		Entity Angular Velocity *3x32 bit integer*
n x 128	Articulation Parameters	Change *16 bit unsigned integer*
		ID-attached to *16 bit unsigned integer*
		Parameter Type *32 bit parameter type record*
		Parameter Value *64 bits*

Table 5: Entity State PDU

was the only one used, shown Table 5 [IST93]. The entity state PDU is generic enough for any entity type, such as an aircraft or human. It is also variable in length so that as the design of DC VET matures, more data can be packed into the PDU packet for transmission over the network.

To articulate *Jack* on other computer stations, status parameters are loaded in the network PDU index of the application and transmitted at desired event criteria. It is possible to pass a PDU every cycle to keep other stations informed about your networked entity's status, but this would quickly saturate the network with a large group of players. To reduce the number of PDUs on the network, data is transferred over the network only when an entity's status has changed or if an event has occurred. When there is no status change, no entity state PDUs are sent for up to five seconds. This helps to maintain a low network load. To determine the network entity's location when no PDU has been received, a dead reckoning algorithm is employed to estimate the entities current location. This feature works quite well, but can cause jumping of the networked *Jack* figure in the world if his speed has changed drastically, or if DC VET is operating on a much slower machine, such as an SGI Indy or Indigo[2].

B. NETWORKING SHORT FALLS

The DC VET has been evaluated using five players and there was no observable lag time or saturation, Figure 12 [ZESW93]. The real-time rendering of the scene with the networked players worked satisfactorily. Users were able to interact with each other and view the changes and effects caused by each other in real-time. A short fall found with networking is the presence of race conditions. If two users perform the same operation at the same time, the last person's action is final. For instance, if one person is closing a door and another is opening it, then the door swings back and forth. This is similar to the reality of two people fighting for the same item. The person who quits first loses, or last
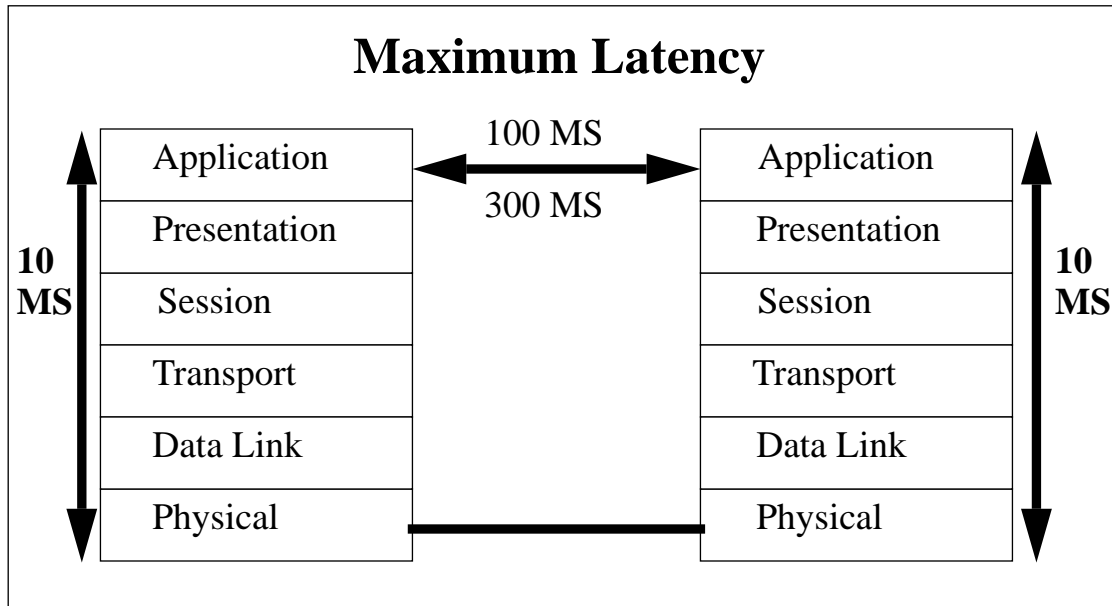
Figure 12: DIS Maximum Latency Specification

wins. With many of the object manipulations in the environment this did not appear to be a major problem. Solutions for undesirable race conditions still require follow on work.

Another potential networking problem, besides race conditions and saturation, is lost or corrupted PDUs. If a PDU is lost and no simulation status has changed, then the dead recognizing algorithm and new PDUs will be able to maintain the integrity of the simulation. However, a networked station will not be able to maintain the simulation if a PDU containing an event such as initializing a fuel oil leak, activation of the halon system, or fire nozzle state change occurs is lost. There would be different states in the same simulation with entities interacting where one has a fire ragging and the other is calm. PDU packet loss or corruption is not common, and this potential networking error has not affected the DC VET application during its nine months of operation to date.

There is another difficulty with new stations entering the DC VET simulation while other networked players have changed states in the simulation. The new player does not receive a previous history of events. This causes doors and values to be in different positions, and damage to bulkheads from fires not to be rendered in the new player's environ-

ment. To solve this, a PDU packet containing the history of the environment would need to be sent out to a new player asking to join the simulation. After receiving this packet, old and new players in the DC VET environment would have the same simulation with the same states. This was not an issue since the DC VET simulation of networked players are together, in the same laboratory or location.

The networking of the environment is another critical factor in creating a virtual environment trainer where multiple people can participate. With the ability to transfer postures and motion of a human, like *Jack*, we can visualize in the virtual environment what other networked players are doing, like Figure 13. Networking has made the Damage
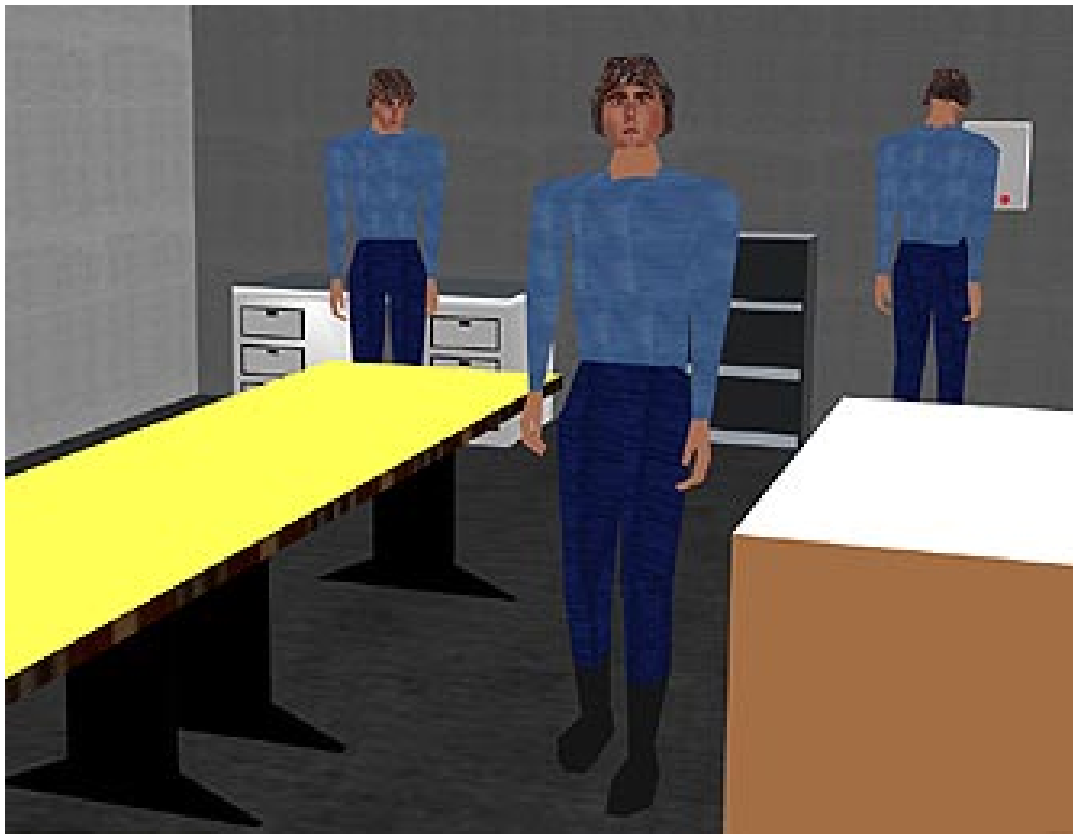
Figure 13: Networked *Jack* Entities

Control Virtual Environment Trainer a platform worth consideration for other types of group training as well.

VI. COLLISION DETECTION

A. INTERSECTION TESTING

1. Deck and Object Collisions

Iris Performer features for computing intersection testing are utilized in the walkthrough project. Intersection testing gives the walkthrough program its interactive capability allowing tests for collision detection, object manipulation and hyper-text displays with sounds. Performer's method tests line segments against the database scene.   In walkthroughs, there are many different objects in the scene which are of different shapes and sizes. Because of the number of objects in a scene, many more line segments are required for intersection testing to resolve collision detection. During the initialization of the program every object is assigned an intersection mask when the model database is loading. Hence, during real-time traversal of the model, there is no delay in learning what each object is. For the training and familiarization phase of this project the intersection masking of objects allows the user to touch any object in the world with the mouse and discover what it is. This is a great tool for the novice sailor who, having never been aboard a ship, needs to know what things are. In the real world, the sailor would have to have a "sea-daddy" follow him around and answer a myriad of questions.

For intersection testing, Performer line segments are taken from the user's view point and are then embedded in a pfSegSet structure that retains the origin, length and direction [KING95]. (The pfSegSet embodies an intersection request as a group of line segments, an intersection mask, discrimination callback and traversal node.) The intersection testing is computed and checked for a bitwise AND of the intersection traversal mask and the intersection mask of the object. When the result is non-zero, an intersection has occurred and the line segment is stored in the above pfSegSet structure which is then tested against the geometry of the object. When an intersection test is positive, information about the intersection point is saved in a Performer pfHit structure for later use by the walk-

through application. Otherwise, there is no intersection and the traversal does not continue checking past the object's child nodes. Within the pfHit structure, information about the intersection's coordinates, normals, transformation matrix of the object, the node name and a pointer to the intersected objects are saved. A query is used to extract the information from the pfHit structure and ,by default, the nearest object to collide with it is the one queried. This can be changed by a discriminator callback function [ROHL94].

There are two types of collision detections tested during each frame in the walk-through environment: deck (ground) collision and object collision. Iris Performer pfSegSet originates from the user's view point. Seven line segments are calculated every frame: one for the deck collision and the other six for object collisions as seen in Figure 14. The deck collision detection scheme points the line segment in the negative Z axis down twenty meters. By checking intersection every frame the height-of-eye is accurately maintained whether it be on a flat surface or an incline, such as a ramp. To prevent walking on objects, such as tables, piping and machinery equipment, the traversal intersection masks are unique for decks. A bitwise AND is computed, and if a DECK_MASK and a COLLIDE_MASK result in a non-zero value, the intersection information is loaded into the pfHit structure. When the user is traversing a ship's ladderwell (stairs), the height-of-eye is adjusted rapidly to the new step which gives the illusion of walking, vice gliding, up or down steps.

Collision into objects is also calculated every frame utilizing six line segments along the X and Y axes in the direction of the user's motion. There are two sets of line segments. The first set is at the origin of the viewer's height-of-eye and projected straight out and 45 degrees from the center, left and right. A second set of segments in the same directions are taken at the user's knee level, which is one-thirds the height-of-eye. Two different levels ensures that the user does not walk into low objects like equipment nor high objects like piping. The length of a line segment is 2.0 times the previous distance between frames or a minimum of 0.1 meters. This prevents the user from passing through an object or bulkhead (wall) before reaching it. When collision detection occurs, the user is returned to the previous location and heading before the collision which gives the user the illusion

Top Down View

Direction of Motion

85°

45°

45°

Line-of-Sight

85°

Axes

Y

X

Object Collision Detection Segments

Side View

Upper Segment Set

Direction of Motion

Lower Segment Set

Axes

Y

Z

Deck Collision Segment
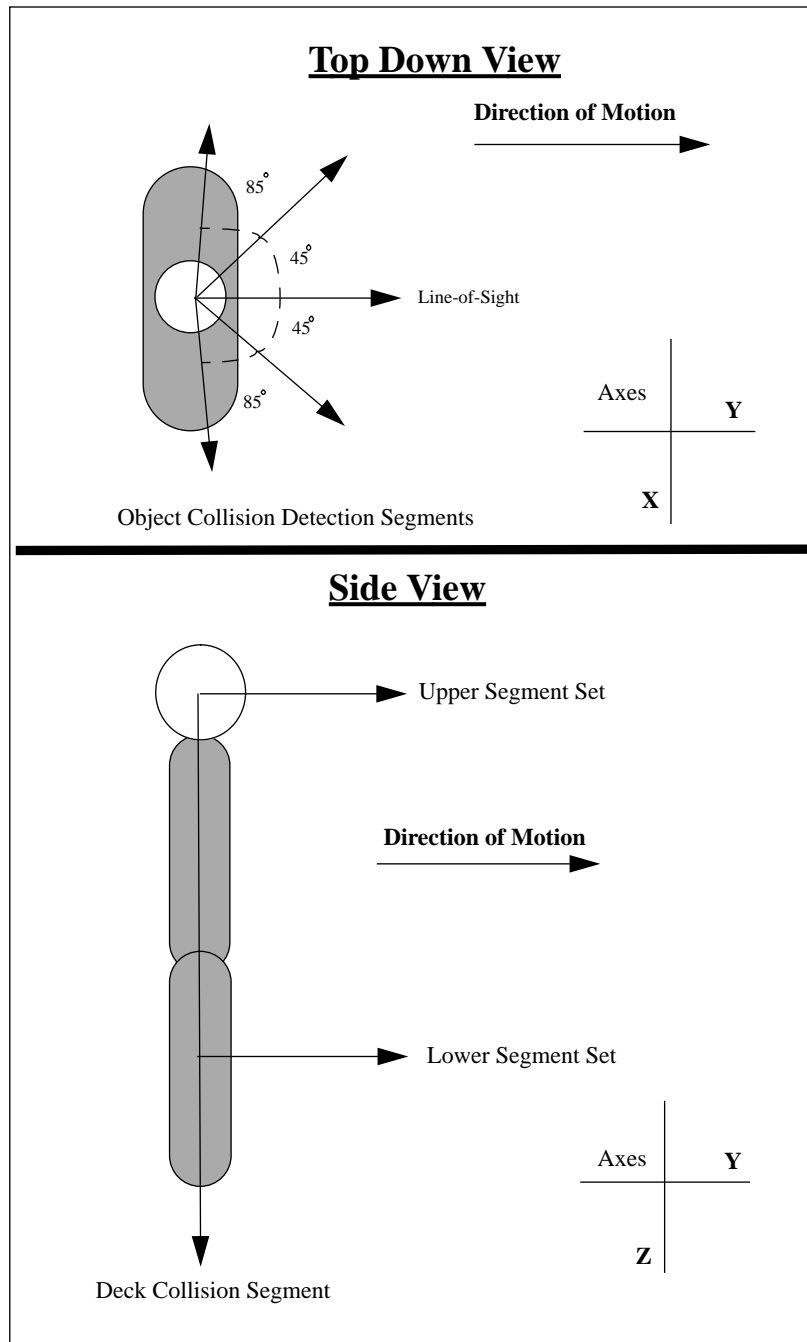
Figure 14: Collision Detection Scheme

of bouncing off the object. When the user is moving backwards, the line segments are simply reversed to perform collision detection in the opposite direction.

The above collision detection method works well most of the time in a non-networked simulation. Note, when the user is networked and able to view others, the above method is limited. When the user is stationary and in close proximity of an object or bulkhead, others can view the networked user's shoulders or hands passing through objects while he rotates when stationary. This occurs only in the networked environment since you cannot see yourself in the stand alone DC VET. To overcome this loss of realism in the networked version, additional line segments are added. Four additional segments, two sets of two, are tested at near right angles to the user's direction at the height-of-eye and knee level. These additional line segments are always .25 meters because, the distance from center of the body to the shoulder is .2 meters. These additional segments function like the previous six line segments. This prevents the user's body from rotating into an object. As the complexity of our human in the world grows the complexity of intersection testing increases. With the articulation of legs and arms with the *Jack*® Motion Library [GRAN94A], dynamic points for intersection testing of the user represented by *Jack* must be incorporated.

Another modification to the collision detection algorithm takes into consideration the user carrying objects in his hands. Previously when carrying objects like a fire hose nozzle, it would pass through other objects or the bulkhead (wall). A solution to solve this problem was to extend the center line segments (upper and lower) when the user is carrying an object. When an object changes state and is placed into the user's hands, an index to the length of the new center line segments are passed to the collision detection scheme. For instance, when the user holds the nozzle, an additional 0.4 meters is added to the center line segments to forecast the collision and prevent the held object from passing through any other objects. Increasing only the center line segments and not the other 8 prevents the user from being trapped in tight locations, like ladderwells or passageways of the model. Adding to the side segments does not represent the user's true occupied space. This solution is not perfect but reduces significantly the chance of viewing user held objects passing through other objects. Another option would be to have a second set of line segments from

the end of the object perform collision detection like the upper or level detection segments show in Figure 15. This could be incorporated into the original collision detection algo-

Top Down View

Direction of Motion

85°

45°

Variable

Line-of-Sight

Length

45°

85°

Nozzle

Axes

Y

X

Object Collision Detection Segments
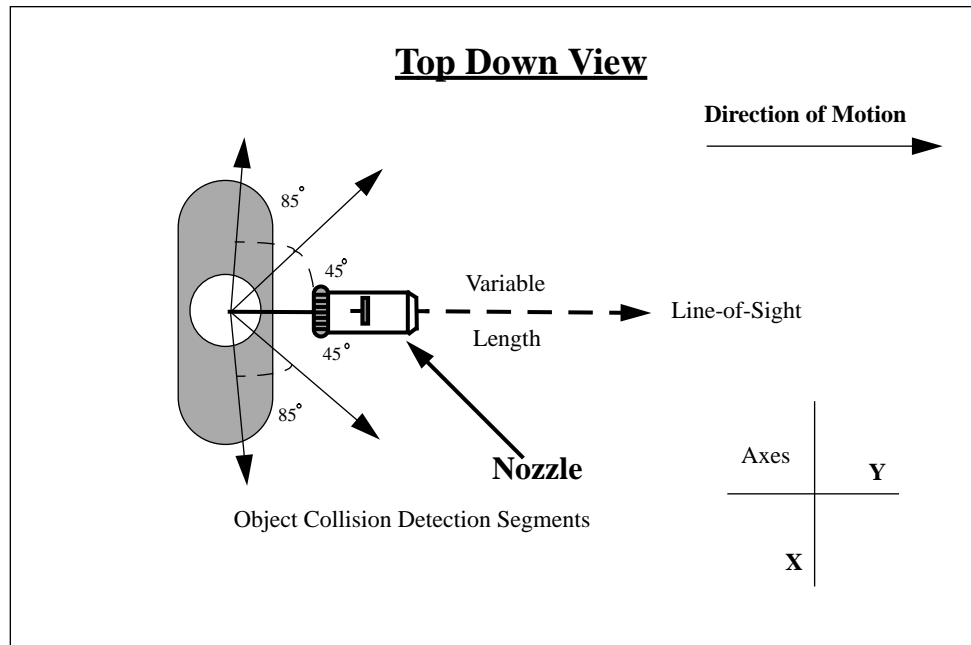
Figure 15: Modified Collision Detection Scheme

rithm or could be a separate collision detection algorithm which returns a boolean value to a positive collision detection. This should reduce the chances even further of any hand held objects passing through other objects.

2. Autonomous Agents

All the conceptual designs of collision detection algorithms discussed so far have been about the user himself in the virtual environment. All collision detection resolutions have been from the user's view point and basically stop the motion of the user and relocate him to the previous location prior to detection. With networked entities this was still a satisfactory means of resolving collisions. However, with the introduction of autonomous agents, such as *Jack*, DC VET needs to determine collisions for its autonomous agents too. The concepts are the same as the previous methods of collision detection, but now each autonomous agent must know if it collides with other objects. A separate function provides

47

this and returns a boolean value indicating collision or not. It is up to the designer to add limited intelligence and have the agents perform in a human like manner when they walk into other players, objects or bulkheads. The Marine outside the Combat Information Center (CIC) in the Antares model walks back and forth until something blocks its path. When the collision occurs, the Marine simply turns around and walks back in the opposite direction. The collision detection can also be used to start events when a user comes in proximity of a room. Sound events can be played or states in the environment can be changed. This is much like a mine waiting to explode when somebody gets close enough.

B. PICKING

Intersection testing is also incorporated into classifying the type of objects in the virtual environment. A Performer function called pfChanPick is used to return the node or object picked. It operates much the same way as intersection testing with pfSegSet and pfHit. When the mouse points to an object, a set of screen coordinates are returned. The pfChanPick function generates a line segment from the user's eye point which passes through the near clipping plane generated by the screen coordinates of the mouse. Results from the intersection testing are stored in a pfHit structure as well. The pfHit structure returns a pointer to the intersected node's object which is then checked against the masks assigned during the program's initialization to see if the object can be manipulated or can display hyper-text with sound. By pressing the mouse buttons while pointing to an object, the user can open doors, turn valves and move objects.

Collision detection is paramount to the success of this project. It aids the user in navigating realistically in the virtual environment and resolves the distinction between objects to manipulate and objects which display information. This interaction helps the immersed user learn the necessary skills required in damage control. In the evaluation of intersection testing for collision detection, object manipulation and activating hyper-text with sounds, there was no noticeable degradation to the real-time rendering of the virtual environment using the Reality Engine 2. However, in the laboratory, using the single

processor Silicon Graphics Indigo^2 Extreme computer and no textures, the frame rate was reduced significantly to five frames/second. As the need for more intersection testing becomes necessary the overall performance of the trainer may decrease. Additional methods may need to be included to increase efficiency and generality of intersection testing.

VII. USER INTERFACE

A. INTERFACE CONTROLS AND DISPLAYS

For the walkthrough project a simple interface was desired because the intended user of the application is not necessarily a computer user. The interface is designed to be fairly intuitive so that there is minimum instruction required. The final design uses a large monitor or a Head Mounted Display (HMD) as the output device, and a mouse as the input device. Data gloves and video tracking devices were considered as the means of tracking the user's motion, but because of the cost or reliability of these items, they were not used. The CAVE (Audio Visual-Experience Automatic Virtual Experience) is another system that would work extremely well in the virtual environment trainer [NEIR92]. However, due to costs and restricted portability, it was not considered for this project. The preferred method for immersion into the virtual world with DC VET is the HMD. The HMD gives the user a wider field of view (100 degrees) and intuitive feeling of navigating the environment. The user just looks in the direction he wants to travel and presses a mouse button to move forward, backward or to stop. If the user wants to open a door, close a valve, press a switch, move an object or inquire about an object, he looks directly at that object in the center of the screen, then presses two mouse buttons together; the object does what the user wants. In spite of its advantages, it is our experience that most people don't like to wear the HMD for extended periods of time. Another problem encountered with the HMD is the difficulty in reading the hyper-text displays. The letters are too small for the 745 by 225 resolution HMD in the laboratory. Better HMDs costing in excess of $25,000 may solve this, but the goal here is a low cost system. With four people networked together for a training scenario, it could get costly to provide better HMDs. Sounds were added for effects and to help alleviate the problem of reading hyper-text with the HMD. When a user inquires about an object, he can hear what the hyper-text displays.

The use of a 1280 by 1024 resolution large screen monitor is just as adequate for displaying the virtual environment.   The monitor is set to a 45 degree field of view which is the normal focusing field of view for humans. Because of the monitor's high resolution, a Graphical User Interface (GUI) is located at the bottom of the scene to assist the user. The user can quickly adjust his height-of-eye, switch from the walk mode (collision detection) to a fly through mode, immediately jump to key points in the ship model, start/stop training and familiarization lessons, or have the trainer show him the way from point A to point B. It also displays the user's current location in the X, Y, Z planes, plus heading, pitch and roll. Another feature is a Deck Overview of the ship which allows the novice user to see an outline of the ship and his current location, represented as a black dot, so he will always know where he is in relation to the ship. Figure 16 shows a scene of the engine room with a hyper-text box displayed, the deck overview and GUI controls.
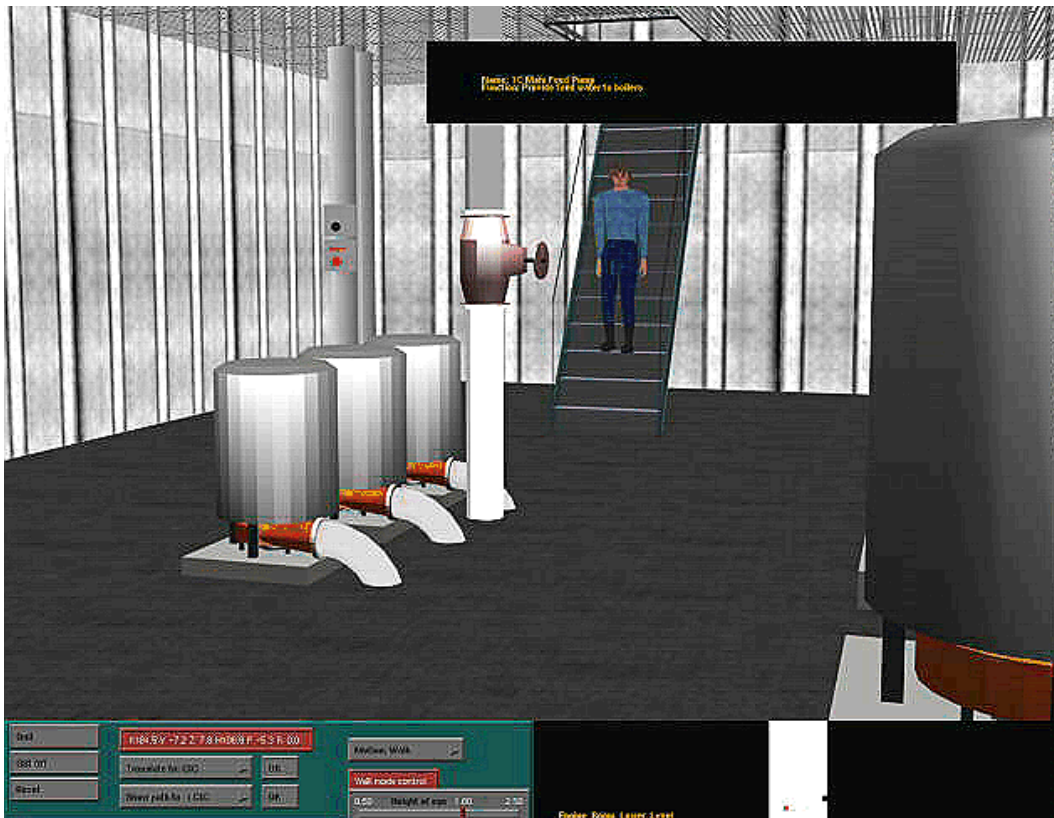
Figure 16: GUI Controls With View of Engineroom

A mouse is used as the primary means of input which controls speed, direction of motion, view point and object manipulation. The operations which can be performed with a mouse are listed in Table 6. To change the user's view point, move the mouse pointer to

Mouse Controls			
Action	**Left**	**Middle**	**Right**
Move Forward	X		
Stop		X	
Move Backwards			X
Close a Door/Valve	X	X	
Open a Door/Valve		X	X
Pick Up a Fire Nozzle		X	X
Start/Stop Fire Nozzle Water		X	X
Start Ventilation System		X	X
Start Halon Flooding System,		X	X
Display/Hear Object Data		X	X
Start/Stop *Jack* Tour Guide		X	X

Table 6: Mouse Button Interface and Functions

one of the four quadrants and the screen will follow. To look up and left, just move the mouse to the upper left quadrant. To move right, move the mouse to the middle right of the screen. When the user wants to open a door, he moves the mouse over the door and presses the right and middle mouse buttons at the same time. There is a "dead zone" in the center two inches of the screen created to prevent the screen from panning in any direction. The keyboard is also used as a means of input and it controls many functions in addition to initiating damage control scenarios. One may use the keyboard as an alternate control instead of the GUI, see Table 7.

The preferred method for immersion into DC VET would be a HMD with a 3D mouse. This would allow the user to gain the sense of reaching out to objects in the environment instead of just pressing a mouse pointer from a 2D mouse over an object at any

Functions	Keyboard Input
Display GUI and Deck Overview	'F1'
Turn On/Off Local Sound Effects	'F2'
Turn On/Off Local & Global Sounds	'F3'
Exit Program	'ESC'
Toggle CPU and Graphics Statistics	'D' or 'd'
Initiate Fire Casualty Sequence	'F' or 'f'
Place Fire Nozzle Back to Fire Station	'P' or 'p'
Toggle Texture Display	'T' or 't'
Toggle Wire Frame Display	'W' or 'w'
Jump to Bridge Way Point	Shift 'B' or 'b'
Jump to CIC Way Point	Shift 'C' or 'c'
Jump to Engineroom Way Point	Shift 'E' or 'e'
Start/Stop Demonstration -- *Jack* as Guide	'J' or 'j'
Start/Stop Demonstration (Camera Follows)	Shift 'J' or 'j'
Start/Stop Demonstration (Wait for User)	'H' or 'h'
Restart Demonstration/Lesson	Shift 'K' or 'k'
Change to New Demonstration/Lesson	'L' or 'l'
Jump to Vehicle Loading Dock Way Point	Shift 'P' or 'p'
Save RGB Image of Display	Control 'PrintScreen'
Move Forward an Event During Demonstration	Shift '+'
Move Backward an Event During Demonstration	Shift '-'

Table 7: Keyboard Interface and Functions

depth. The use of voice recognition for issuing commands would have also been ideal for this type of environment. Voice recognition would allow the user to walk around the ship without having to focus on objects to be manipulated, or requiring a coordinated mouse click to initiate a response. Voice as a input device would also allow the user to make quicker responses to the environment. Another reason for voice activated commands with DC VET is that all the engineering scenarios can only be activated by the keyboard. The

software to test voice recognition was not available for this thesis, but is a worthy consideration for future work.

Another alternative to voice recognition and the keyboard as input devices is to use floating icons that can be seen in the HMD or from the console. These icons would become visible to the user when a mouse button is pressed. An icon menu could appear and lead to sub icon menu's to prevent the screen from being cluttered. When using the HMD and icon menus, a user can start an engineering casualty easily. The choice of icons would have to have some likeness in representation to the action taken, so the user would need to instinctively know what the icon means, especially if the user is new to DC VET. The use of icons with a 2D mouse is feasible with the DC VET application and should be incorporated into the next revision.

B. SOUNDS AND EFFECTS

The addition of sounds into DC VET was necessary to increase the feeling of immersion. Originally sound-bites would be used to indicate a response to inform the user of general conditions. The user would hear an alarm sound when there was an engineering casualty, or be able to tell that the ventilation system was activated for de-smoking. Sounds provide instant feedback and allow the user recognition of events that are not visibly apparent. With the addition of sound-bites for recognition of events, came the need for the ability to give information about the world. In the original DC VET, the user was able to navigate through the ship and read text which explained what an object was and its basic function. Now with the addition of the sound library, the user can hear and read the description simultaneously.

To implement real time sounds in DC VET, a sound library was utilized. Sound-bites that are used repeatedly, like clanking noise of feet on a ladderwell (stairs), ventilation noise, or alarms, are loaded into memory during program load up. This allows for instant audio feedback with no noticeable delay. However, lengthy sounds that are describing what objects are in the world are not loaded into memory, and have a slight latency

when playing the sound file. With DC VET being a networked environment came about the desire to networked sounds. Also, not all the reality engine graphic computers have sound capability. A sound server program to play these sounds was necessary. The DIS communication protocol, used for networking multiple stations, was also utilized for the networking of sounds. The sound server is capable of providing sounds for just one station or a group of stations in the same area. The sound server application reads PDU packets off the network, waits for a sound request and plays the requested sound file. When operating with multiple stations, it is also verifies if a sound request for the same sound was just requested. This prevents hearing the repeated playing of sounds, like alarms, multiple times because more than one station has requested it at nearly the same time. If the sound server did not discriminate requests, the sounds will have a reverb effect. This occurs because the sound server is capable of playing up to four sounds simultaneously and can play the same sound file at the same time.

Most sounds in DC VET are initiated because an event, engineering casualty, or an input from the user was sensed by the program. When the user causes an event to occur, a call to play the corresponding sound-bite is sent to the sound server. For audio to explain what objects are in the world, additional callbacks were used to find the corresponding sound file of objects created in the model. MultiGen allows for the imbedding of text into the model. This allows the designer to write information about the object's name and function, and have the corresponding sound file be saved with the object. When the user presses his mouse button to learn about an object, the name of the sound file is retrieved from the model database and is sent to the sound server. The sound server receives the request and plays the information sound file, and the user hears about what the item in question is within a second.

Sounds can be also used when creating scripted training lessons using the autonomous *Jack*. The script can call sound files to inform the user of the purpose of this lesson or about things that can not be truly experienced in the world. The designer has the ability to add sounds effects to the beginning of each event planned in the scripted lesson. This

allows for the flexibility of using audio when needed during the designing of a *Jack* lesson in the environment.

C. SUMMARY

The use of sound for information and effects in DC VET adds to the realism of the environment in much the same way NPSNET has done for years. DC VET was created more for familiarization training, vice simulation skills enhancement, and has the ability to instruct using audio. The use of audio is just like having a tour guide or an instructor teaching you about the world. Sounds improve the DC VET from the original design. To go one step further, the incorporation of short video clips could also be used to teach skills that are difficult to render in the existing virtual environment. For instance, DC VET could teach a person how to don an OBA (Oxygen Breathing Apparatus) by just clicking a mouse button and mouse pointer on an OBA and watching a short video of donning the OBA. There are many means of adding training into the virtual environment. The more training that can be imbedded into the virtual environment, the more it will be capable of augmenting existing training and replacing other training methods.

VIII. CONCLUSION

A. RESULTS

The goal of this thesis is to increase the capability of the DC VET (Damage Control Virtual Environment Trainer), to prove the feasibility of using virtual environments as a trainer for the U.S. Navy. To achieve this, the thesis explored several areas of virtual reality technologies, and achieved the following results:

- The simulation is networked so that several users can interact in the same virtual environment for training. Users see networked articulated humans and interact with each other. Also the network allows users to hear sounds on non-sound capable reality engine computers by playing sounds on remote sound capable computers.

- The collision detection mechanism is enhanced to prevent the user from moving through decks, bulkheads and other objects.

- Scripted training scenarios are created to both test and train the user. The scenarios can be reviewed repeatedly until the user feels proficient in the subject area.

- Sounds are added to increase the feeling of immersion. Sounds are heard by the users to indicate events occurring, give instant feedback and instruct users about the environment.

- Several training devices are incorporated into the simulation. These include a hypertext window with sound to display information, scripted lessons which teach the user navigation skills, and an articulated human to guide the participant through the environment.

B. RECOMMENDATIONS FOR FUTURE WORK

There are still several areas which need to be implemented or enhanced before this trainer is ready to be used by the fleet. They are listed in the order of significance.

1. Create a Naval Ship Model From Actual Ship Data

This thesis is limited in familiarization training until DC VET is simulating an actual naval vessel. Since an actual naval vessel is too large to model without using CAD data,

a method to convert existing CAD data into a format which can be visualized is a necessity. The solution is made more difficult by the fact that contractors use their own proprietary CAD software to design ships. The navy needs to create a standard format for visualization data and require that all CAD data delivered from contractors be implemented in that format. Perhaps this could be a standard for NGCR (Next Generation Computer Resources) SPARWAR (Space And Naval Warfare Systems Command) to undertake in the near future [NGCR95].

2. *Jack*® **Motion Library**

The implementation of the *Jack*® Motion Library should be linked to the NPSNET *Jack* Motion Library to take advantage of the improvement and upgrades of *Jack* ML. Currently, DC VET uses its own separate version with different articulation methods and features. One standard would eliminate redundancy and add more functionality. The existing postures and motion for *Jack* are limited to soldier movements and need to be expanded to include normal every day postures of humans. Addition hand motions, created in DC VET for *Jack,* need to be developed and all *Jack* ML features unique to DC VET need to be incorporated into NPSNET.

3. **Networking Larger Virtual Environments**

The existing network system handles the database and required number of participants quite well. However, the method used to transmit the database information cannot be expanded to a significantly larger database, such as a model of an entire ship. To solve this problem, additional PDU types, listed in Table 4, besides the entity state PDU need to be incorporated as the complexity of DC VET grows. Additional testing needs to be done to determine the best method to update a large-scale database with a large number of networked participants.

4. Improved Interface and Input Devices

To maximize use of the HMD version of DC VET, a better interface must be implemented. The existing configuration of the NPS Graphics and Video Laboratory HMD makes it difficult for the user to examine objects which are "behind" his initial orientation; this is most noticeable when the user attempts to reverse his path. In addition, it is difficult for the user to interact with a three dimensional world using a two dimensional mouse device. Perhaps a three dimensional input device, such as a 3-D mouse or data glove, would allow the user to manipulate objects in a more realistic manner. This would allow the user the capability to travel in a direction other than his view direction and greatly improving the realism of DC VET. Also, the use of voice recognition, as a primary means to start events, control articulated humans and learn about objects, would increase the ease of interaction and possibly improve the rate of learning.

5. Dynamic Casualty Control Scenarios

There are currently only three casualties which can be simulated in DC VET, and they can only occur at fixed locations in the ship model. Having such limited scenarios makes it easy for the crew to spot only one or two indications and know exactly what the drill is and how to fight it optimally. To make training more realistic, dynamic conditions that change every scenario but maintain the desired training effect are essential. Variability must be incorporated, otherwise the participants learn only how to combat just a specific type of casualty and not the whole gambit.

6. Merge into NPSNET

The concepts in the design of this Damage Control Virtual Environment Trainer can be incorporated in the Naval Postgraduate School's Networked Virtual World (NPSNET). In its next phase, it is planned to have people and vehicles depart or board a ship from the NPSNET terrain database. The fundamentals of walkthrough design implemented in this project will be applied to buildings so soldiers can enter and move about in them realistically as well. Using both the implementation of PVS and Collision Detection

within confined areas in NPSNET, a soldier will be capable of going into buildings in the existing simulation. These modifications will add a new level of simulation training. Soldiers can learn familiarization of specific buildings ahead of time, so that they can maneuver quickly and smartly during a small arms battle. Much like a naval vessel, soldiers can improve team interaction and learn the surroundings for future missions before experiencing the real world.

7. Increased Data Display

Currently, when the user selects an object, only the name and function of the object with background sounds are displayed. This capability should be expanded to display a wide range of information, such as what system the object is in, what is its normal position, what effect manipulating it will have on the ship, etc. In the case of complex systems, DC VET could be capable of calling up and displaying the diagram of the system for the user to immediately learn about how each object fits into the larger picture. Until virtual environments are capable of displaying the more intrinsical details of the objects in the world, the embedding of information, such as video clips, should be explored. With video clips, detailed illustrations of equipment operation and implementation can be readily shown. For example, the user can learn how to don an OBA or fix a damaged part on a Gas Turbine Engine. Also, the use of icons and a transparent HUD (Heads up Display) can help the user interact more easily within the environment and make the decision process easier.

8. Testing and Evaluation of Virtual Environment Trainers

DC VET was created to train sailors at sea, and needs to be scientifically verified in that it can effectively train sailors. An objective evaluation, considering human factors and recording empirical data, is required to prove if VE is a viable solution to training. The Operations Research and System Management Departments of the Naval Postgraduate School should create experiments which can measure the training efficiency of the simulation.

9. More Realistic and Efficient Collision Detection

The existing method of using line segments for collision detection and picking should be improved by a more efficient volume intersection algorithm. This would reduce the amount of overhead involved in the simulation. Also, the algorithm should be modified so that when the user hits an object he doesn't just stop, but instead simulates bouncing off at the angle of reflection.

C. FINAL REMARKS

This thesis research created a prototype virtual reality shipboard trainer which can be feasibly used as a trainer. In achieving this large task the project examined the construction of a large model to train with and built a simulation that is networkable to anyone with an Internet connection. DC VET employed the basic concepts of efficiently rendering the scene in real-time by using the methods of PVS and LOD. User interaction within the simulation was improved by incorporating the means of collision detection to navigate realistically around the world and to manipulate objects. Training scenarios were created with environmental effects (smoke and fire) for more realism. The design of an intuitive GUI for use with a monitor and the HMD increased the feeling of immersion. Sounds have been integrated to indicate events and instruct the user within the environment. The incorporation of the *Jack*® Motion Library into DC VET adds more realism of human representation in the virtual environment. To assist in navigation and training, an articulated human guides the user through the ship model. The ability to train a sailor in familiarization of simple tasks in damage control is feasible. The concepts here are transferable to other types of training and are ready to be adopted.

APPENDIX A.   USER'S GUIDE

This appendix is the user's guide for operating the DC VET (Damage Control Virtual Environment Trainer) [KING95].   It explains starting and running the system and discusses the interface options and various commands available to interact with the virtual ship. The guide includes the combination of new and old DC VET versions.

A. STARTING DC VET

The DC VET can be run on a variety of SGI graphics platforms. During the initialization process it determines the number of processors available on the platform and configures the multi-processing mode of the application.

To start the DC VET, change to the directory in which the executable "walk" file is located. The executable file "walk" is located in the "/workd/obyrne/OByrneThesis/code" directory at the Naval Postgraduate School Graphics and Video Laboratory. By simply typing "walk" followed by a return, the program begins execution of a non-networked, standard monitor display DC VET.

To network DC VET or direct the visual output to a head-mounted display, command line options are used following the "walk" command. See Table A-1 for a quick reference of command line options.   To join an exercise in progress with other

Line Command	Option
-h	Output to HMD
-n	Network Simulation
-s	Play sounds locally
-z	Network Observer Mode

Table A-1: Command Line Options

workstations, the **-n** command line option is required. To observe a networked simulation but not participate, **-z** command line option is used. If directing the visual output to a head-

mounted display, the **-h** command line option is required. To play sounds locally, use the **-s** command line option.

The program takes approximately three minutes to complete the initialization phase. During the first portion of this period, the models and textures used for the simulation are loaded. Once loaded, a title screen consisting of the title of the project and its authors is displayed on the screen until the application is finished initializing (approximately twenty seconds). Following application initialization, the textures loaded earlier are downloaded into random access memory in order that they can be quickly accessed when needed. The textures are displayed on the screen as they are being downloaded. Once the texture download is completed the application begins and places the user in the Combat Information Center.

B. PROGRAM TERMINATION

There are two methods to exit the DC VET. One method is to depress 'Esc', or the shell interrupt key, typically 'Cntrl-C'. The other method is to select the quit menu button on the graphical user interface (GUI). Both of these options completely shuts down the system including any processes spawned during the application. (Note: the GUI option is not available when wearing the HMD)

C. SCREEN LAYOUT

The standard screen layout takes up the entire screen and includes the virtual scene display, graphical user interface (GUI) and deck overview. A pop-up window, which displays information about objects in the virtual ship, is displayed when objects are selected with the mouse. These displays and their relative locations on the screen are shown in Figure A-1 and Figure A-2.

The virtual scene display takes up ninety percent of the screen. The overview display and GUI can be turned off to allow the full screen to be taken up by the virtual scene display by either depressing 'F1' on the keyboard or selecting "GUI off" on the GUI. To re-enable the GUI and deck overview display, 'F1' must be depressed on the keyboard.

1. Deck Overview

The deck overview channel is located on the lower right hand portion of the screen as shown in Figures A-1 and A-2. It provides an overhead view of the deck on which the user is presently located. The deck lay-out is graphically displayed in two dimensions showing the locations of ladders, bulkheads, doorways and passageways. A black position cursor shows the user's position in the virtual ship and moves as the user moves along the deck in the virtual environment.

```
┌─────────────────────────────────────────┐
│                                          │
│        ┌────────────────────────┐        │
│       ╱                          ╲       │
│      ┌────────────────────────────┐      │
│      │                            │      │
│      │       ┌──────────────┐     │      │
│      │       │   POP-UP     │     │      │
│      │       │ INFORMATION  │     │      │
│      │       │   DISPLAY    │     │      │
│      │       └──────────────┘     │      │
│      │                            │      │
│      │         SCENE              │      │
│      │         DISPLAY            │      │
│      │                            │      │
│      │                            │      │
│      ├──────────────────┬─────────┤      │
│      │                  │  DECK   │      │
│      │ GUI MENU BUTTONS │ OVERVIEW│      │
│      └──────────────────┴─────────┘      │
│                                          │
└─────────────────────────────────────────┘
```
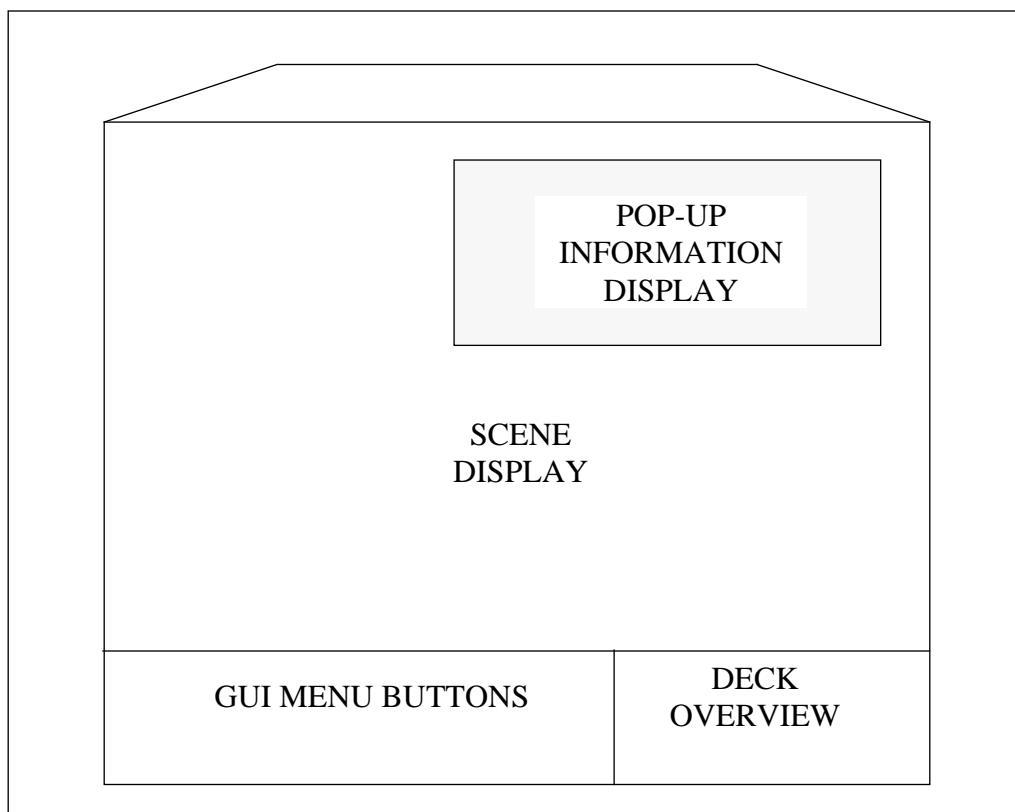
Figure A-1: Monitor Display

2. Pop-Up Data Display Window

When the user selects an object with the mouse, a pop-up window containing information about the object selected is displayed in the upper right hand corner of the
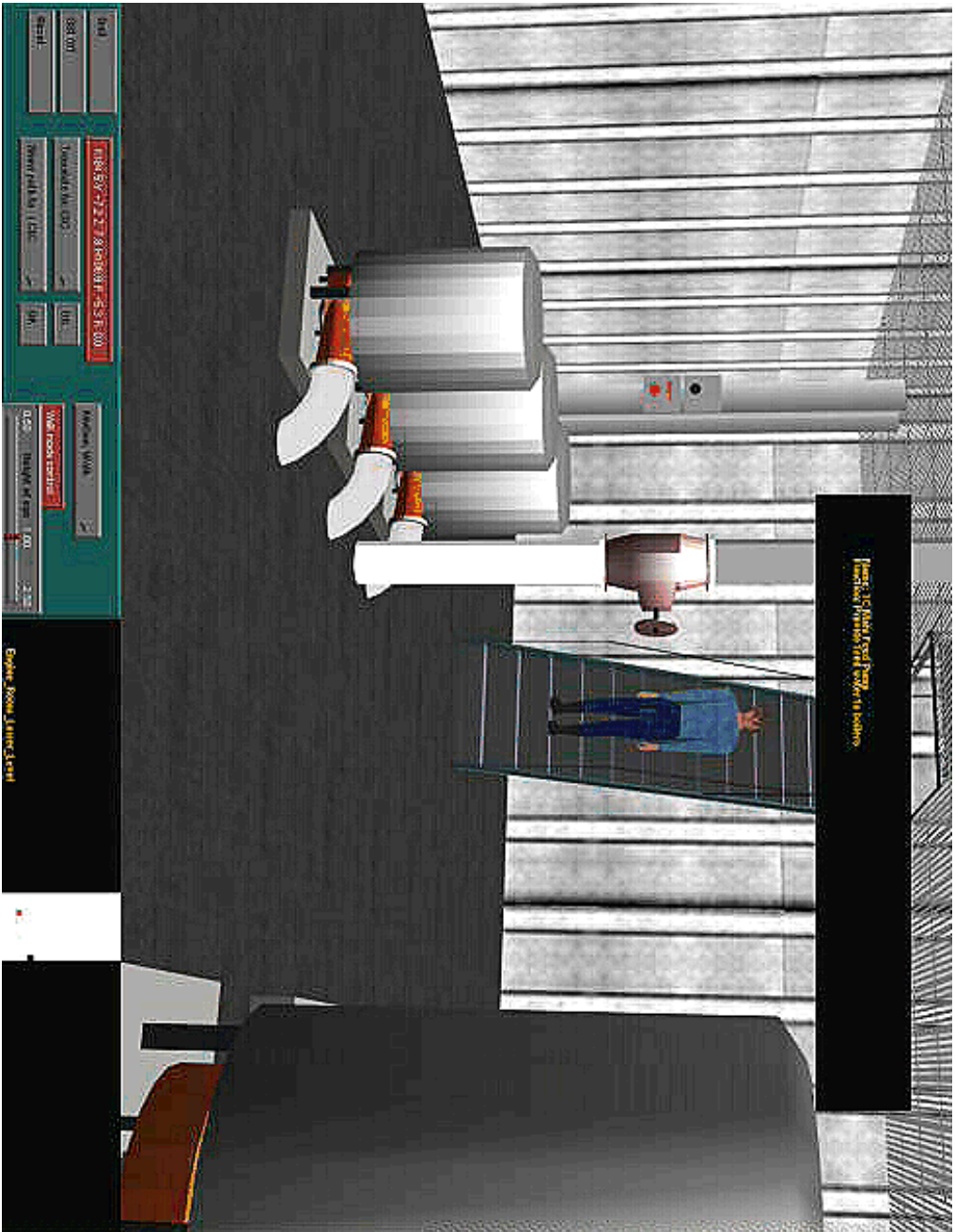
Figure A-2: DC VET Console Display

screen as shown in Figure A-2. The display stays on the screen until the mouse buttons are released. Also, when sounds are activated the user can hear an audio sound-bite about the object in question. (Not all objects have sounds associated with them.)

3. Graphical User Interface

The graphical user interface (GUI) provides the user with an "easy to use" menu interface to perform an assortment of functions. The GUI is located on the lower left corner of the screen as shown in Figure A-1. A representation of the GUI is displayed in Figure-A-2.

D. OPERATION

There are two DC VET operating modes. The first mode, "walk", which is the default mode, simulates naturally walking through the virtual ship. Collision detection is enabled meaning that the user cannot walk through objects. The other mode is "fly" which enables the user to move through the ship as if he was flying. In "fly" mode, collision detection is disabled allowing the user to fly through objects. These modes are changeable by the "Mode" menu toggle button on the GUI.

1. Mouse Operations

Natural walking ("walk" mode) or flying ("fly" mode) is simulated with the aid of a mouse. By depressing either the right mouse key (forward motion) or the left mouse key (reverse motion), the user gains speed and moves through the environment in the direction the user is looking. The middle mouse button causes the viewer to stop.

The direction the user is looking is also determined by the mouse. The view direction changes in the relative direction that the mouse cursor is, positioned from the center of the screen. For example, the farther to the right of center the mouse cursor is the quicker the individual will turn to his right. The range of motion in the vertical direction is capped to straight up (+90 degrees) and straight down (-90 degrees) when in the "walk"

mode. There is a one inch box in the middle of the screen referred to as the "dead zone" in which the mouse cursor, if inside this area, does not cause the view direction to change.

The mouse is also used to select objects in the virtual ship for either object data display, manipulation or movement. To select an object, the user places the mouse cursor on an object and depresses the middle and either the left or right mouse button at the same time. If the object is not a movable object, a pop-up window is displayed in the upper right hand corner of the screen as shown in Figure A-2 for as long as the mouse buttons are pressed down.

a. Objects Which Move

All doors throughout the ship and cabinet covers in the Radar Room can be opened and closed. To open a door, the right and middle mouse button must be depressed at the same time with the mouse pointing to the door. The door rotates in its open direction until it reaches its maximum rotation of ninety degrees or until the mouse buttons are released. To close the door, the left and middle mouse buttons are depressed at the same time and the opposite motion occurs.

Two valves located in the Engineroom Lower Level are both capable of being opened and closed. The operation of valves is similar to the operation of doors as far as the method used to open and shut the valves. When opening a valve the valve stem rises and the valve hand-wheel rotates in a counter-clockwise motion; the opposite occurs when shutting the valve.

A vari-nozzle, when picked, is moved from its storage location in Engineroom Lower Level to directly in front of the user's view at belt level. The vari-nozzle can be opened and shut once the user has the nozzle in front of him by further clicking the mouse on the nozzle. The nozzle is moved back to its storage location by depressing 'p' or 'P' on the keyboard.

b. Objects Which Can Be Manipulated

A ventilation fan controller and halon activation system controller in the Engineroom Lower Level are capable of being turned on and off by the mouse. To manipulate the controllers, the controller button must be picked as described above. Mouse control functions are provided in Table A-2.

Mouse Controls			
Action	**Left**	**Middle**	**Right**
Move Forward	X		
Stop		X	
Move Backwards			X
Close a Door/Valve	X	X	
Open a Door/Valve		X	X
Pick Up a Fire Nozzle		X	X
Start/Stop Fire Nozzle Water		X	X
Start Ventilation System		X	X
Start Halon Flooding System,		X	X
Display/Hear Object Data		X	X
Start/Stop *Jack* Tour Guide		X	X

Table A-2: Mouse Button Interface and Functions

2. Graphical User Interface (GUI)

The GUI, displayed in Figure A-2, provides the following functions, starting in the upper right corner and proceeding clockwise:

a. Quit Button

The quit button causes the user to leave the application. Depressing the 'Esc' key also accomplishes the same function.
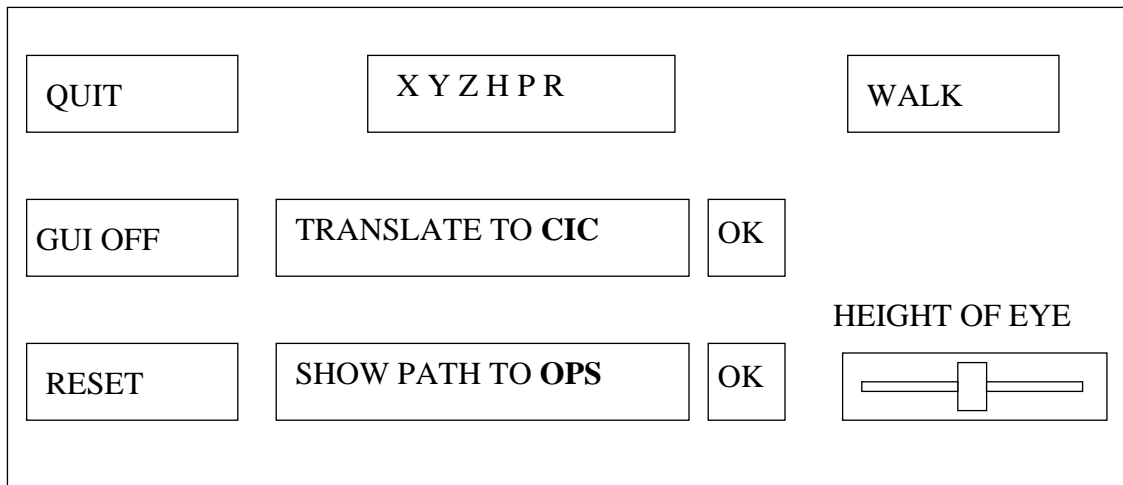
Figure A-3: Graphical User Interface

b. User's Position Display

The user's location in three space is displayed here as X, Y, and Z coordinates. Also, the user's heading, pitch and roll are also displayed.

c. Traversal Mode Selection

The two modes of operation, "fly" and "walk", which were previously discussed, are controlled by this toggle menu button.

d. Height of Eye Control

The height of eye control slider enables the user to vary the eye point height above the deck while in the "walk" mode in order that objects which are close to the deck or up high can be viewed at a closer distance. The user can vary height between 0.5 - 2.5 meters.

e. Path Planning Selection

A path planning tool is provided which takes the user along a path from his present location to a location of his choosing via the optimal route at normal walking speed. The locations which can be selected include CIC (Combat Information Center), the Radar

Room, the Operations' Office, DCC (Damage Control Central), the Hull Technician's Shop and the ladder to the engineroom. The user selects his destination by clicking the button titled "Show path to: <destination>." As he clicks it, a different destination is displayed. When the desired destination is displayed, the user selects the "OK" button next to it and begins travelling to that destination. At this point, the menu button changes to "Stop walking to: <destination>", and if the user selects it, he is no longer transiting to the destination and regains control of his own motion.

f. Reset Button

This button allows the user to reset the application to its original state. All objects are returned to their initial position, all casualties are terminated, any damage caused by casualties is repaired, valves are opened and the atmosphere is cleared. (Note: this change in state is not networked and will cause networked simulations to have different scenes.)

g. Toggle GUI Button

The "GUI-Off" menu button turns the GUI and the deck overview off providing more screen display for the scene. The GUI and deck overview can be returned to the screen display by depressing 'F1' on the keyboard.

h. Translation Selection

To allow the user the ability to quickly "jump" from one location to another in the virtual ship, a translate menu button is provided. Preset anchor points to key locations are embedded in the software code. These locations include CIC, DCC, Engineroom Lower Level, Bridge and the Vehicle Loading Deck.

3. Keyboard Operations

The keyboard is primarily used to initiate casualties in the virtual environment. It also provides another method to accomplish some of the functions which are provided by the GUI. The keyboard inputs and their functions are listed in Table A-3.

Functions	Keyboard Input
Display GUI and Deck Overview	'F1'
Turn On/Off Local Sound Effects	'F2'
Turn On/Off Local & Global Sounds	'F3'
Exit Program	'ESC'
Toggle CPU and Graphics Statistics	'D' or 'd'
Initiate Fire Casualty Sequence	'F' or 'f'
Place Fire Nozzle Back to Fire Station	'P' or 'p'
Toggle Texture Display	'T' or 't'
Toggle Wire Frame Display	'W' or 'w'
Jump to Bridge Way Point	Shift 'B' or 'b'
Jump to CIC Way Point	Shift 'C' or 'c'
Jump to Engineroom Way Point	Shift 'E' or 'e'
Start/Stop Demonstration -- *Jack* as Guide	'J' or 'j'
Start/Stop Demonstration (Camera Follows)	Shift 'J' or 'j'
Start/Stop Demonstration (Wait for User)	'H' or 'h'
Restart Demonstration/Lesson	Shift 'K' or 'k'
Change to New Demonstration/Lesson	'L' or 'l'
Jump to Vehicle Loading Dock Way Point	Shift 'P' or 'p'
Save RGB Image of Display	Control 'PrintScreen'
Move Forward an Event During Demonstration	Shift '+'
Move Backward an Event During Demonstration	Shift '-'

Table A-3: Keyboard Interface and Functions

4. Head-mounted Display Operation

The program is configured to run with a head-mounted display (HMD) if the **-h** command line option, discussed previously, is used. The configuration changes the window size and graphics video format to be compatible with the HMD requirements. "Walk" mode is the only mode of operation available when wearing an HMD. The GUI, deck overview and pop-up window are also not displayed with the HMD.

Walking through the virtual ship when wearing an HMD is very similar to the walking method discussed previously. The only difference lies in the method in which the view direction is determined when wearing an HMD. The HMD's tracking device translates the HMD's direction of view to an appropriate view direction in the virtual environment. Therefore, to walk around the virtual ship, the user physically looks in the desired direction and depresses the appropriate mouse buttons.

Movable objects such as doors, valves, vari-nozzle and manipulated objects such as fan and halon controllers can still be picked while wearing an HMD. The method is similar to the mouse picking method discussed previously, however, instead of selecting objects with the mouse pointer, objects are selected by placing the cross-hairs in the center of the HMD view on the object.

E. CASUALTY SCENARIOS

1. Fire Casualty Sequence

By depressing either the 'f' or 'F' keys on the keyboard, the main space fire casualty sequence commences with a JP-5 fuel oil leak at a piping elbow joint in the lower level of the engineroom. The fuel oil leak develops into an engineroom fire if the oil leak is not stopped within twenty seconds by shutting an isolation valve upstream of the leak. The fire breaks out with a radius of two meters and, if no extinguishing agent is applied, grows with each frame cycle until it reaches a maximum radius of 3.5 meters.

The fire can be extinguished by either obtaining and opening a vari-nozzle to apply high velocity spray to the base of the fire or activating the halon fire extinguishing system. If using the vari-nozzle to apply the extinguishing agent, the firefighter must be within six meters of the fire and apply the high velocity spray within five degrees of either side of the fire's origin. The fire decreases in radius at a rate commensurate with the amount of time the extinguishing agent is applied. If networked and more than one individual is putting out the fire, the fire goes down more quickly. If the firefighter does not keep the high velocity spray within the above constraints, the fire will grow as before. If instead, the

firefighter activates the halon fire suppression system to extinguish the fire, the fire responds as in reality and decreases at a quicker rate than if water is applied.

Once the fire is initiated, the environment in the main space begins to fill up with smoke. Gray-black smoke incrementally fills the compartment for as long as the fire continues to burn causing a reduction in visibility until a minimum visibility of five meters is reached. Once the fire is out, the smoke can be cleared by turning on ventilation fans.

2. Steam Leak Casualty

By depressing either the 's' or 'S' key on the keyboard, a steam leak develops at a union on the deaerating feed tank (DFT) outlet piping just below the DFT feed isolation valve. The size of the steam leak changes if the DFT feed isolation valve is manipulated. As the valve handwheel is closed, the leak reduces in proportion to the percentage the valve is opened. Once the DFT feed isolation valve is fully shut and ventilation fans are activated, the steam can be dissipated.

The steam leak also causes the atmosphere to become obscured as in the fire casualty discussed above. The difference is that the color of the obscurity for steam is white-gray, compared to the grey-black of the fire.

F. TRAINING LESSONS

1. Operating Lessons

There are three modes of operation to choose from with scripted training lessons. In all the modes *Jack* acts as a guide or instructor who leads the user through the ship. In the first mode, *Jack* moves about the ship, as scripted in the lesson, on his own until the script is completed. The user can follow *Jack* or interact with the environment, but *Jack* will not wait. To start and pause this demonstration mode, depress either the 'j' or 'J' key on the keyboard. The second mode is the same as the first except *Jack* will wait for the user to follow him. If the user gets more than four meters away, *Jack* stops and waits. This allows the user to follow the *Jack* guide and pause while learning what other items of

importance. To start and pause this wait and demonstrate mode, depress the shift and either 'j' or 'J' keys on the keyboard. The last mode is a view mode where the scene changes and follows *Jack* through the lesson. The user does not interact with the environment in this mode, but just watches events occur. This is much like a video presentation. To start and pause this video demonstrational mode, depress either the 'k' or 'K' key on the keyboard.

The user can switch from any of the three demonstration modes at anytime by depressing the key associated with the mode twice. To pause the demonstration, depress any of the three keys depressed which started the demonstration. To continue where the demonstration was paused, just depress the key associated with the desired demonstration mode. To restart a demonstration, from the beginning, depress the shift and either the 'k' or 'K' keys on the keyboard and *Jack* guide will be moved to the original location and start the lesson over. To move ahead during the demonstration or to fast-forward through events, depress the shift and '+' keys the respective number of times. To repeat an event or go back several events, depress the shift and '-' keys the respective number of times.

There can be up to five different scripted training lessons ready for execution with DC VET. These lessons are stored in a data file which is loaded into the computer memory just before execution. The default scripted lesson ready for presentation is the file "scriptData0.dat". To change to another prepared scripted lesson depress the either the 'l' or 'L' keys on the keyboard. This will cycle the lessons available up to the last, lesson four (index is 0 to 4), and after four, it will repeat over with lesson zero.

2. Creating Scripted Training Lessons

A data file is utilized to read the scripted lessons created by the user. No re-compiling of the application is necessary to swap lessons. DC VET is capable of scanning for up to five different lessons which are stored in files "scriptData0.dat" through "scriptData4.dat". These files are located in "/workd/obyrne/OByrneThesis/code/models" directory. Using any standard text editor, actions are written or edited using the format provided in Table A-4. An example data file is provided in Figure A-4. Information to

move *Jack* through the environment and have him perform actions listed in Table A-5 are recorded in these files. These index numbers are also located in "blueJack.h" file in the "workd/obyrne/OByrneThesis/code" directory.

The Reference Listing for Each Column's Data															
Event Sequence Number	*Jack* Scripted Command	Location X Coordinate	Location Y Coordinate	Location Z Coordinate	Final Heading	Head's Pitch	Speed / Rate of Turn	Object Type	Miscellaneous Value	Sound Reference Number	Camera's View Location X Coordinate	Camera's View Location X Coordinate	Camera's View Location X Coordinate	Camera's Direction of View	Camera's Elevation of View

Table A-4: Script Data File Format

Scripted *Jack* Events	
Event Number	**Event Requested**
0	Load Next Event into the Ready Queue
1	Move to Location
2	Pivot head to Elevation
3	Go Up/Down Stairs
4	Open/Close Doors
5	Open/Close Valves
6	Perform Hand Signals
7	Start Fuel Oil Leak Then Fire Casualty
8	Pick Up/Place Down Nozzle -- Turn Water On/Off
9	Start Steam Leak Casualty
10	Activate Ventilation
11	Activate Halon Fire Suppression
12	Play Sound-bites
13	Jump to Other Sections of the Ship
14	Wait/Pause *Jack* at Current Location
15	Display Hyper Text Box
16	Exit Scripted Lesson
17	Restart Lesson From the Beginning

Table A-5: Autonomous *Jack* Instructions

When designing the scripted lesson, just move about the ship as the *Jack* guide and record the events that are to happen. Include the X, Y, Z coordinates and the heading and pitch desired at that point. To determine a camera view, the easiest process is to watch *Jack* move once a scripted lesson is recorded to the file. Play the lesson back and look through each event for where the optimal location for observing *Jack* is. Record those coordinates as well and play them back. The user does not have to quit DC VET, but simply pause the scene and open up a text editor to record the changes. When the lesson is switched or the

```
0  1  182.0  -1.4 21.39 090.0  0.0  0.2   0 0 0  187.3 -0.4 23.1  090.0  -12.0
1  2  182.0  -1.4 21.39 270.0  0.0 25.0   0 0 31 187.3 -0.4 23.1  090.0  -12.0
2  1  186.5  -0.5 21.39 272.0  0.0  0.3   0 0 0  187.3 -0.4 23.1  090.0  -12.0
3  4  186.5  -0.5 21.39 272.0  0.0  3.0   5 1 0  182.3 -2.2 23.1  291.0  -13.5
4  1  188.5   0.0 21.39 277.0 -10.0 0.4   0 0 0  182.3 -2.2 23.1  291.0  -13.5
5  2  188.5   0.0 21.39 352.0  80.0 25.0  0 0 0  182.3 -2.2 23.1  291.0  -13.5
6  1  189.0   5.9 21.39 352.0  10.0 0.4   0 0 0  191.2 -5.4 23.1  014.0  -9.0
7  1  191.0  -1.1 21.39 180.0 -25.0 0.3   0 0 0  191.2 -5.4 23.1  014.0  -9.0
8  3  191.0  -3.5 18.05 180.0 -30.0 0.8   0 0 25 1912 -10.4 19.8  011.0  -4.5
9  1  188.0  -3.5 18.05 000.0  0.0  0.3   0 0 0  187.1 2.6 19.8   011.0  -4.5
10 2  188.0  -3.5 18.05 120.0 10.0 15.0   0 0 0  184.2 -6.2 19.8  304.7  -6.0
```

Figure A-4: Example "scriptData.dat" File Contents

same lesson is restarted, the "scriptData.dat" file is re-read into memory. This allows ease of creating scripted files and is less time consuming for development. Note there is a limit of 50 events per scripted lesson for a total of 250 events. If the user needs more than this to conduct training, DC VET can easily be modified to accommodate more scripted lesson files or increase the number of events for each lesson.

G. SOUND SERVER

1. Staring the Sound Server Program

The executable file for the sound server program is located in the "/workd/obyrne/ OByrneThesis/code/SoundServer" directory at the Naval Postgraduate School Graphics and Video Laboratory. By simply typing "soundServer" followed by a return, the program begins execution of a networked receiver which plays sounds requests from DC VET networked stations. When the program is ready to play sounds the user will hear, "Sound Server Activated". This application is a simple C program to allow Indy and Indigo systems in the laboratory, that do not have enough memory, to run a sound server program

with a graphical interface. A second version of the sound server has a Motif GUI interface for simple operations. With the GUI version you can temporarily turn off sounds or play sounds from a particular host. The GUI sound server program is executed by typing "GUIsoundServer" at the command line.

2. Networking Sounds

To network a user's simulation to the sound server program, the user must be in the network mode. When operating in the network mode, the Entity State PDU carries the index for sound files to be played, if any, and the sound server extracts the index from the PDU and plays the corresponding sound file.

3. Local Sound Capability

If the system running DC VET is sound capable, sounds do not have to be sent to a sound server to be heard. The user at his local station can play sounds created by the application. To play sounds locally, the **-s** option is required after typing "walk" at the command line. This option will not transmit sounds to a sound server. When using the network mode, the user will have the sound events sent to the sound server and still can hear their sounds on their own system.

4. Features

The sound server can read PDU packets from all the stations on the networked simulation. It has to discriminate sound requests and not play the same exact sounds at the same time. If this happened a reverberated sound effect occurs and the sound is not clearly audible. A queue is used to store sound request to ensure the same sound is not played at the same time. The queue is also used to ensure that a sound request is not lost. The typical SGI sound capable computer can play four sounds simultaneously. Beyond that the sound server waits and then plays the request. If the sound server is overloaded, some sounds may not occur in real time. It is also possible to overflow the queue and lose sounds requests.

However, this has not occurred in practice since the buffer is large enough to handle a robust training simulation.

The user also has the choice whether not to play his sounds locally or send them over the network. There are two options for sounds, local and global. Local sounds are sound effects such as bumping into a wall or opening a door. Global sounds are announcements over the 1MC (also known as PA, Public Announcement) like fire in the engineroom or the sound of an explosion. The user can turn on/off local sounds by depressing the 'F2' key and global sounds with the 'F3' key. By default these sounds are active when the simulation starts.

5. Sound Files

Sounds are recorded and saved to the "/workd/obyrne/OByrneThesis/code/sounds" directory. The sound server program is capable of playing up to 45 different sound requests. These .aiff or .aifc sound files are loaded into the memory of the sound server computer. This allows real time play back of sounds corresponding to events and requests from the given simulation. The names and paths for the sounds to be loaded are recorded in a data file called "sounds.dat". This is also located in the "SoundServer" directory. Associated names for sounds such as GQ for the general quarters sound or MFP1A for a definition of a main feed pump are listed in the file "sounds.h". This file can be found in the "/workd/obyrne/OByrneThesis/code" directory. For writing scripts, use the index number corresponding to the sound file from "sounds.dat". An example of "sounds.dat" is provided as Figure A-5. To edit sounds and save them to an .aiff or .aifc file, use the SGI "soundeditor" program. "soundeditor" is a GUI program that is fairly intuitive to use and allows for easy mixing or splicing of sounds.

```
0   /workd/obyrne/OByrneThesis/code/sounds/NPSsounds/sounds/ak47.aiff
1   /workd/obyrne/OByrneThesis/code/sounds/NPSsounds/sounds/aaahhhhh.aiff
2   /workd/obyrne/OByrneThesis/code/sounds/bump.aiff
3   /workd/obyrne/OByrneThesis/code/sounds/NPSsounds/sounds/explosion1.aiff
4   /workd/obyrne/OByrneThesis/code/sounds/NPSsounds/sounds/explosion2.aiff
5   /workd/obyrne/OByrneThesis/code/sounds/stairs.aifc
6   /workd/obyrne/OByrneThesis/code/sounds/NPSsounds/sounds/soundServer.aiff
7   /workd/obyrne/OByrneThesis/code/sounds/MFP1A.aifc
8   /workd/obyrne/OByrneThesis/code/sounds/halon.aiff
9   /workd/obyrne/OByrneThesis/code/sounds/GQ.aiff
10  /workd/obyrne/OByrneThesis/code/sounds/MFP1A.aiff
11  ........................................ through  ...........................................................41
42  /workd/obyrne/OByrneThesis/code/sounds/follow_me.aiff
43  /workd/obyrne/OByrneThesis/code/sounds/vent_on.aiff
44  /workd/obyrne/OByrneThesis/code/sounds/water_spray.aiff
```

Figure A-5: Example "sounds.dat" File Contents

LIST OF REFERENCES

[ADAM95]   Adams, Nina and Lang, Laura, "VR Improves Motorola Training Program", AI Expert, Volume 10, Number 5, May 1995, pp 13.

[AIRE90]   Airey, John M., Rohlf, John H. and Brooks, Fredrick P. Jr., "Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments", Department of Computer Science, University of North Carolina Chapel Hill, Computer Graphics, Volume. 24, Number 2, March 1990, pp 41.

[BROO92]   Brooks, Frederick P. Jr., "Final Technical Report - Walkthrough Project June 1992 to Computer and Information Science and Engineering National Science Foundation", UNC Technical Report 92-026, June 1992.

[BUKO95]   Bukowski, Richard W. and Sequin, Carlo H., "Object Associations: A Simple and Practical Approach to Virtual 3D Manipulation", Proceedings of the 1995 Symposium on Interactive 3D Graphics, April 1995, pp 131-138.

[CATM84]   Catmull, Ed, Carpenter, Loren, and Cook, Rob, "Private and Public Communications", 1984.

[DOD92]   Department of Defense, "Defense Modeling and Simulation Initiative", Washington D.C., May 1992.

[ERTE95]   Ertel, Lawrence R., "Birth of a New Machine: Making of Impact", IRIS Universe, Number 32, 1995, pp 16-23.

[FUNK94]   Funkhouser, Thomas A., Khorramabadi, Delnaz, Sequin, Carlo H., and Teller, Seth J., "UCB System for Interactive Visualization of Large Architectural Models", April 13th, 1994.

[GRAN94A]   Granieri, John P., and Badler, Norman I., "Simulating Humans in VR", Center for Human Modeling and Simulation University of Pennsylvania, October 12th, 1994.

[GRAN94B] Granieri, John P., "Jack/TTES: A System for Production and Real-time Playback of Human Figure Motion in a DIS Environment", Center for Human Modeling and Simulation University of Pennsylvania, August 5th, 1994.

[IST93] Institute for Simulation and Training, "Communication Architecture for Distributed Interactive Simulations (CADIS)", Final Draft Proposed IEEE Standard, IST-CR-93-20, University of Central Florida, June 28th, 1993.

[KING95] King, Tony E., and McDowell Perry L., "A Networked Virtual Environment For Shipboard Training", Master's Thesis, Naval Postgraduate School, March 1995.

[LAST95] Lastra, Anselmo, Molnar, Steven, Olano, Marc, and Wang, Yulan, "Real-Time Programmable Shading," Proceedings of the 1995 Symposium on Interactive 3D Graphics, April 1995, pp 59-66.

[LOCK94] Locke, John, "An Introduction to the Internet Networking Environment and SIMNET/DIS", Computer Science Department, Naval Postgraduate School, October 24th, 1994.

[LUEB95] Luebke, David and Georges, Chris, "Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets", Proceedings of the 1995 Symposium on Interactive 3D Graphics, April 1995, pp 105-106.

[MACE95] Macedonia, Michael, R., Brutzman Donald P., Zyda, Michael J., Pratt, David R.,Barham Paul T., Falby John S., and Locke, John, "NPSNET: A Muti-Player 3D Virtual Environment Over The Internet," Proceeding of the 1995 Symposium on Interactive 3D Graphics, April 1995, pp 93-94.

[NATI94] National Academy of Sciences National Research Council Committee on Virtual Reality Research and Development, "Report on the State-of-the-Art in Computer Technology for the Generation of Virtual Environments", January 1994.

[NCGR95] SPAWAR "NGCR Acquisition Guide", Next Generation Computer Resources Document No. AST 001 Ver. 0.11, March 30th, 1995.

[NEIR92] Cruz-Neira, Carolina, Sandin, Daniel J., DeFanti Thomas A., Kenyon, Robert V., Hart, John C., "The Cave: Audio-Visual Experience Automatic Virtual Environment", Communications of ACM, Volume 35, Number 6, June 1992.

[NRL95]   Naval Research Laboratory, "Virtual Environments for Shipboard Damage Control and Firefighting Research", http://nrl.com.damageControl, 1995.

[ROHL94]   Rohlf, John and Helman, James, "IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics", 1994 SIGGRAPH Course Notes, May 1994.

[TELL91]   Teller, Seth J. and Sequin, Carlo H., "Visibility Preprocessing For Interactive Walkthroughs", Computer Graphics, Volume 25, Number 4, July 1991, pp 61.

[ZESW93]   Zeskwitz, Steven R.,"NPSNET: Integration of Distributed Interactive Simulation (DIS) Protocol for Communication Architecture and Information Exchange", Master's Thesis, Naval Postgraduate School, January 1994.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   Cameron Station
   Alexandria, VA   22304-6145

2. Dudley Knox Library 2
   Code 052
   Naval Postgraduate School
   Monterey, CA   93943-1501

3. Chairman, Code CS 2
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA   93943

4. John S. Falby, Code CS/FA 2
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA   93943

5. Dr Michael J. Zyda, Code CS/ZK 10
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA   93943

6. LT James E. O'Byrne 2
   149 Beach 92nd Street
   Rockaway Beach, NY   11693

7. Dr Bernard Ulozas 2
   Naval Personnel Research and Development Center
   53335 Ryne Road
   San Diego, CA   92152-7250